Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Teja Cetinski

Primerjava funkcionalnosti in zmogljivosti sistemov za upravljanje s podatkovnimi bazami MySQL in MariaDB

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matjaž Kukar

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Odprtokodni sistem za upravljanje s podatkovnimi bazami (SUPB) MySQL je brez dvoma najpopularnejši relacijski SUPB. Vendar pa so se s prehodom pod okrilje podjetja Oracle pričeli pojavljati dvomi o njegovi razvojni viziji, zato so se začele pojavljati binarno združljive, a čisto odprtokodne izpeljanke. Med njimi je najpopularnejša MariaDB, ki je že precej razširjena, ponuja pa tudi nekatere nove, uporabne funkcionalnosti. Kandidatka naj v svojem delu razišče trditve o združljivosti MySQL in MariaDB in kvantitativno ovrednoti zmogljivost obeh sistemov s serijo testov na ekvivalentnih strojnih platformah. Sistematično naj primerja funkcionalnosti, ter preizkusi in ovrednoti dodatne mehanizme, ki jih ponuja MariaDB. Na koncu naj poda zaključne ugotovitve o tem, kaj pridobimo ali izgubimo z uporabo enega ali drugega sistema.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Teja Cetinski, z vpisno številko **63070049**, sem avtorica diplomskega dela z naslovom:

Primerjava funkcionalnosti in zmogljivosti sistemov za upravljanje s podatkovnimi bazami MySQL in MariaDB

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, 2015 Podpis avtorja:

Zahvaljujem se mentorju, doc. dr. Matjažu Kukarju, za usmerjanje in pomoč pri izdelavi diplomske naloge.
Hvala sošolcem za vse izmenjane ideje, nasvete in zapiske ter prijateljem, ki so šli z mano skozi vzpone in padce študija. Hvala tudi sodelavcem na Arnesu, zaradi katerih je bila izvedba praktičnega dela naloge znatno olajšana.
Neskončno se zahvaljujem svoji družini; Kaji za sestrsko podporo, staršem pa za vso finančno in moralno spodbudo.
In, ne nazadnje; hvala, Matej – za vse.

Mojim dragim staršem.

Kazalo

Seznam uporabljenih kratic

kratica	**angleško**	**slovensko**
RDBMS	relational database management system	sistem za upravljanje relacijskih podatkovnih baz
SUPB	database management system	sistem za upravljanje podatkovnih baz
OLTP	on-line transaction processing	sprotno obdelovanje transakcij
ACID	atomicity, consistency, isolation, durability	atomarnost, konsistentnost, izolacija, trajnost
MVCC	multi-version concurrency control	nadzor nad sočasno uporabo z več različicami
API	application programming interface	programski vmesnik
CRUD	create, read, update, delete	ustvari, beri, posodobi, izbriši

Povzetek

MySQL je najbolj priljubljen odprtokodni sistem za upravljanje z relacijskimi bazami. Z Oraclovim prevzemom podjetja Sun in s tem MySQL so se pojavili dvomi o nadaljnjem razvoju sistema in ohranjanju njegove odprtokodnosti, kar je povzročilo pojav izpeljank sistema MySQL, ki slednjega neposredno nadomeščajo ter so vedno bolj priljubljene. V okviru diplomske naloge smo zato raziskali združljivost, funkcionalnosti in performančne izboljšave najpopularnejše od njih, MariaDB. V prvem delu naloge smo na kratko predstavili MySQL in njegove izpeljanke, v nadaljevanju pa podrobneje opisali zmožnosti MariaDB in jih primerjali z zmožnostmi MySQL. V drugem delu smo predstavili testno okolje, orodje za izvedbo zmogljivostnih testov in teste, s katerimi smo merili performanse obeh SUPB. Dobljene rezultate smo grafično prikazali in analizirali ter ugotovili, da je performančno MySQL novejše različice precej izboljšan, vendar pri večini izvedenih operacij vodi MariaDB, kar je v skladu z navedbami razvijalcev. Prav tako MariaDB ponuja nekaj zelo uporabnih zmožnosti, kakršnih pri MySQL zaenkrat še ni moč najti.

Ključne besede: MySQL, MariaDB, zmogljivost, funkcionalnost, testi.

Abstract

MySQL is the most popular open-source relational database management system. Oracle's acquisition of Sun and thereby of MySQL raised doubts about the vision of development, causing the appearance of forks, drop-in replacements of MySQL, which are becoming increasingly popular. In this thesis we therefore explored compatibility, features, and performance enhancements of the most popular fork MariaDB. In the first part we briefly presented MySQL and its forks, then described in more detail MariaDB's features and compare them to MySQL's. In the second part we presented the test environment, the benchmark tool, and the tests that were used to measure both DBMSs' performance. We then visualized and analyzed the benchmark results, concluding that while newer version of MySQL is considerably improved, most operations are carried out faster in MariaDB, which is in compliance with the developers' statements. Furthermore, MariaDB offers several highly useful features that MySQL does not yet include.

Keywords: MySQL, MariaDB, performance, functionality, tests.

Poglavje 1

Uvod

MySQL je po nekaterih podatkih najpogosteje uporabljan odprtokodni sistem za upravljanje z relacijskimi podatkovnimi bazami (RDBMS) [1], upoštevaje tudi lastniške, pa je po razširjenosti na drugem mestu, takoj za sistemom Oracle [2]. Ob Oraclovem prevzemu MySQL-a se je skupina razvijalcev odcepila in ustvarila svoj SUPB, MariaDB, ki ohranja odprtokodnost in povezanost s skupnostjo.

Z naraščanjem priljubljenosti sistema MariaDB, ob seznamih njegovih izboljšav, nenazadnje pa ob opuščanju sistema MySQL s strani velikanov, kot je Google, se porodi vprašanje, kaj (in če) pridobimo z ne več tako novim MariaDB in ali je zamenjava na mestu. Zanimala sta nas tako performančni vidik kot obseg in uporabnost funkcionalnosti, ki jih sistem ponuja.

Diplomsko delo je tako vsebinsko razdeljeno na dva dela, ki se ponekod prekrivata. Na začetku se dotaknemo zgodovine sistema MySQL ter nastanka njegovih izpeljank, predvsem sistema MariaDB (poglavje 2). Nato pregledamo glavne podobnosti in razlike med sistemoma ter raziščemo obstoječe in nove zmožnosti in izboljšave obeh ter jih primerjamo in opišemo v poglavju 3. Oba sistema smo namestili in ju poganjamo vzporedno na dveh navideznih strojih; raziskane koncepte tako hkrati s prebiranjem obsežne dokumentacije tudi neposredno preizkusimo in nekatere ponazarjamo s kratkimi primeri.

V drugem, praktičnem delu diplomske naloge predstavimo izbrano orodje za izvajanje primerjalnih testov zmogljivosti ter opišemo predpripravljene in lastne teste, s katerimi smo merili čas izvedbe različnih operacij na enem in drugem sistemu (poglavje 4). Na koncu predstavimo njihove rezultate (poglavje 5), zaključimo pa

s sklepnimi ugotovitvami v poglavju 6.

Diplomska naloga vsebuje tudi dodatek z izvorno kodo po meri spisanih testov in drugih skript, ki smo jih uporabili pri testiranju.

Poglavje 2

Pregled razvoja odprtokodnega SUPB MySQL

2.1 Zgodovina

Sistem MySQL je produkt švedskega podjetja MySQL AB. Razvit je bil v sredini 90-ih let iz lahkega RDBMS mSQL, prvotno za osebno rabo, danes pa je priljubljena izbira za spletne strani, platforme in aplikacije. V letih do 2005 sta verzijam 3.x sledili verziji 4.0 in 5.0, v okviru katerih je sistem pridobil pomembne zmožnosti, kot so gnezdene poizvedbe, pripravljeni stavki in transakcije.

Leta 2008 je podjetje MySQL AB prevzelo podjetje Sun Microsystems, tega pa je leta 2010 kupila korporacija Oracle. Zaskrbljen, da bo v dobiček usmerjeni Oracle pri razvoju dal prednost poslovni (*enterprise*), plačljivi različici in bo brezplačna, odprtokodna tako zastala, je eden od ustanoviteljev in osrednjih razvijalcev MySQL-a, Michael "Monty" Widenius, ustvaril izpeljanko (angl. *fork*) MySQL-a 5.1, poimenovano MariaDB, ki ostaja odprtokodna in v rokah skupnosti [3]. Prva beta-različica je izšla oktobra 2009, prva stabilna pa sledečega februarja [4].

Poleg zagotavljanja združljivosti z MySQL – MariaDB naj bi bila t. i. zamenjava *drop-in* slednjega, torej jo lahko le "vstavimo" na MySQL-ovo mesto in deluje – je osnovno vodilo MariaDB ohraniti zanesljiv, robusten in skalabilen SUPB, ki hkrati funkcionalno in varnostno izboljšuje MySQL ter ostaja resnično odprtokoden.

Med prvimi, ki so MySQL zamenjali za MariaDB, so bili Mozilla ter Linuxovi distribuciji Fedora in openSUSE v začetku 2013. Sledila sta Arch Linux in Wikipedia, septembra istega leta pa še Google. Kot privzet strežnik MySQL jo imajo med drugim distribucije Red Hat Enterprise, Slackware in Centos, v mnogih drugih pa je na voljo v repozitorijih programske opreme [5] [6] [7].

2.2 Izpeljanke

Negotovost o prihodnosti sistema MySQL naj bi se v manjši meri pojavila že ob Sunovem prevzemu podjetja, ob prihodu Oracla pa so se ugibanja in strahovi le še okrepili. Izpeljankam, ki so bile razvite že prej za specifično uporabo (npr. Googlova) ali so v glavnem vključevale programske popravke/vstavke (OurDelta, Proven Scaling), so se pridružili večji, odmevnejši projekti – samostojni sistemi, ki so nastali bodisi iz potreb po dodatnih zmožnostih bodisi kot posledica nestrinjanja z Oraclovo politiko delovanja [8] [9].

Percona Server

Podjetje Percona se ukvarja z razvojem programske opreme za MySQL, ki med drugim vključuje rešitvi za varnostno kopiranje podatkovnih baz in postavitev gruče MySQL, skupek orodij za upravljanje MySQL, vtičnik za nadzor idr., za nas najbolj zanimiv pa je Percona Server, ki je opisan kot izboljšana zamenjava/alternativa (prav tako *drop-in*) za MySQL in obljublja preboj v zmogljivosti, skalabilnosti in funkcionalnosti [10].

Percona Server temelji na shranjevalnem mehanizmu XtraDB (ta je izboljšana izpeljanka in zamenjava *drop-in* shranjevalnega mehanizma InnoDB), optimiziran pa je za računalništvo v oblaku, dostop NoSQL in shrambo na sodobni strojni opremi (bliskovnih pomnilnikih).

Percona Server je tako kot MariaDB osnovan na (enako oštevilčenih) različicah MySQL, vsebuje pa dodatne zmožnosti, ki jih je sicer moč najti le v poslovni različici MySQL, npr. zmogljivo varnostno kopiranje (z orodjem XtraBackup), napredno gručenje (z XtraDB cluster), visoko varnost z vtičnikom za avtentikacijo PAM (*pluggable authentication modules*, mehanizem, ki omogoča fleksibilno modularno avtentikacijo), bazen niti za doseganje visokih performans in napredno

diagnostiko (funkcionalnost, ki jo najdemo tudi v MariaDB) [11].

Sistem Percona Server naj bi uporabljalo na tisoče velikih podjetij, med katerimi zasledimo velike ponudnike spletnega gostovanja, med bolj znanimi pa Opera, Flickr, Tumblr, Scribd, Wikia in StumbleUpon [12].

Drizzle

Projekt Drizzle je tako kot MariaDB izpeljanka sistema MySQL, in sicer različice 6.0. Ta je bila v razvoju v letih 2007 do 2009, ko so izdali zadnjo alfa-različico te veje. Funkcionalnosti, ki so bile implementirane v 6.0, so bile ali pa še bodo vključene v kasnejše izdaje, tj. 5.5, 5.6 in 5.7.

Drizzle je odprtokoden sistem, napisan v programskem jeziku C++ in razvit za Unixu podobne sisteme, namenjen in optimiziran pa predvsem za spletne storitve in oblačno infrastrukturo. Razvoj sega v leto 2008, prva splošno dostopna izdaja (angl. *generally available release*) pa je izšla marca 2011.

Drizzle naj bi bil okleščen odvečne kode, velik del preostanka pa je spisan na novo in zasnovan v obliki vtičnikov. Zaradi takšne arhitekture ter razlik v sintaksi in podatkovnih tipih sistem ni zamenjava *drop-in* za MySQL, je pa prehod s slednjega možen in zanj ponujajo orodje [13]. Poleg tega so za Drizzle na voljo knjižnice za gonilnik JDBC ter za odjemalca PHP in C/C++, podpira pa ga tudi priljubljen grafični vmesnik phpMyAdmin [14]. Projekt je nekoliko zamrl in stabilne izdaje po septembru 2012 ni bilo več.

MariaDB

Michael Widenius je zaradi nezadovoljstva s Sunovo razvojno politiko MySQL zapustil že pred Oraclovim prevzemom. S še nekaterimi drugimi razvijalci Marie, shranjevalnega mehanizma, ki naj bi bil vključen v prihodnje različice sistema MySQL, naj bi ustvarili aktivno skupnost in vejo "MySQL Maria", ki bi zagotavljala hitrejši in preprostejši MySQL z manj hrošči, predvsem pa popolno odprtokodnost, ki bi omogočala tudi prispevke zunanjih sodelavcev [15].

Ob najavi Oraclovega prevzema je Widenius pozval k njegovi preprečitvi in s tem "reševanju MySQL", prepričan, da MySQL-a v Oraclovih rokah ne čaka svetla prihodnost [16]. Ko je bil prevzem vendarle potrjen, so Widenius in nekaj drugih

osrednjih razvijalcev MySQL ustvarili izpeljanko in se lotili projekta MariaDB.

Leta 2012 je bila ustanovljena fundacija MariaDB, ki pospešuje in skrbi za razvoj sistema MariaDB in njegovo skupnost [17]. Podjetje Monty Program, ki ga je Michael Widenius ustanovil po odhodu iz podjetja Sun, se je leta 2013 združilo s SkySQL, podjetjem za svetovanje in podporo za MySQL, to pa se je oktobra 2014 preimenovalo v MariaDB Corporation. Po Wideniusovih besedah se bo podjetje zaradi neposredne povezanosti z imenom tako zagotovo posvečalo razvoju sistema MariaDB in mu zagotovilo potrebne resurse zanj [18].

Poglavje 3

Primerjava sistemov MySQL in MariaDB

MariaDB izhaja iz kode MySQL, iz katerega je bila ustvarjena veja oziroma izpeljanka. Enaki so vsi ukazi, vmesniki, knjižnice, API-ji, nameščene datoteke in njihove lokacije, konfiguracijske datoteke (z izjemo delov, ki jih definira MariaDB), vrata in vtičnice [19]; namesto paketa MySQL lahko torej namestimo paket MariaDB (iste različice) in sprememba ne bi smela biti opazna. Na primer, strežnik/storitev na Linuxu v obeh primerih zaženemo s `service mysql start`, v ukazno lupino se v obeh primerih prijavimo z `mysql -u uporabnik -p`.

3.1 Številčenje MariaDB

Osnovno pravilo pri orientaciji po številkah različic je, da MariaDB vsebuje vse oziroma vsaj tisto, kar ima enako oštevilčen MySQL. Prva verzija MariaDB, 5.1, izhaja iz MySQL 5.1, iz nje pa naslednici 5.2 in 5.3 [20], ki verzijo 5.1 dopolnjujeta z novimi funkcionalnostmi in izboljšujeta njeno zmogljivost (različic 5.2 in 5.3 v MySQL sicer ni). Zadnja vsebuje tudi dele kode iz MySQL 6.0, ki (še) ni bil uradno izdan.

Naslednja različica, 5.5, je postavljena na osnovi združitve prejšnje (5.3), MySQL 5.5 in nekaj dodatnih funkcionalnosti. Ta verzija vsebuje shranjevalna mehanizma XtraDB, ki je vključen v samo jedro in privzet, in InnoDB, ki je na voljo kot

vtičnik.

Zaradi velikega preskoka v obsegu funkcionalnosti oziroma dejstva, da so pri MariaDB postavili svoje temelje in ne sledijo več (strogo) MySQL, je verzija sistema MariaDB, ki sledi prejšnji, "poimenovana" 10.0. Razvijalci zaradi prej navedenega menijo, da oštevilčenje 5.6 ne bi bilo ustrezno (oziroma bi bilo celo zavajajoče) [21]. MariaDB 10.0 tako nadaljuje od 5.5, vključuje nekatere komponente iz MySQL 5.6 in dodaja mnogo novih. Naslednja različica, MariaDB 10.1, ki je še v razvoju, pa vključuje tudi zmožnosti iz MySQL 5.7 [22].

3.2 Hiter pregled

Kot že rečeno, sta binarna vmesnika MySQL in MariaDB v osnovi enaka – *drop-in* zamenjave drugače niti ne bi bilo možno zagotoviti. Razlike se pojavijo pri sistemskih spremenljivkah kot posledica drugačne implementacije obstoječih (npr. bazena niti ali XtraDB namesto InnoDB) ali pojava novih funkcionalnosti (npr. shranjevalni mehanizem Aria v MariaDB) – spremenljivke imajo tako drugačne vrednosti ali pa sploh ne obstajajo. Iz istega razloga MariaDB porabi nekoliko več pomnilnika, ki pa ga lahko sprostimo s spreminjanjem vrednosti parametrov, npr. velikosti predpomnilnika MyISAM/Aria [23] [24].

V okviru diplomskega dela smo raziskali in primerjali zadnji enako oštevilčeni različici obeh sistemov, tj. MySQL 5.5.40 in MariaDB 5.5.40, ter najnovejši splošno dostopni izdaji, tj. MySQL 5.6.22 in MariaDB 10.0.15.

Navedimo najprej nekaj izboljšav in dodatnih zmožnosti, ki jih ponuja MariaDB 5.x v primerjavi z MySQL 5.x [25]:

- novi shranjevalni mehanizmi Aria, XtraDB, PBXT (do 5.5), FederatedX, OQGRAPH, SphinxSE, TokuDB,
- navidezni in dinamični stolpci,
- razširjena uporabniška statistika,
- NoSQL-vtičnik HandlerSocket,
- poročanje o napredku operacij `ALTER TABLE` in `LOAD DATA INFILE`,
- segmentiran predpomnilnik ključev za MyISAM,
- hitrejše poizvedbe z združitvami, gnezdene poizvedbe, pretvorbe naborov znakov in izračuni kontrolne vsote tabel,

- odpravljeni hrošči in opozorila ter predloženi testni primeri zanje in, nenazadnje,
- odprtost vse kode (tudi zaprtih modulov iz MySQL).

Novosti, ki jih izpostavlja Oracle v MySQL 5.6 [26]:

- dostop NoSQL z uporabo API-ja Memcached in InnoDB,
- izboljšani InnoDB – izboljšane performanse in prepustnost, sprotne operacije, polnotekstovni indeksi,
- izboljšani optimizator – optimizacija gnezdenih poizvedb, boljša diagnostika (ukaz `EXPLAIN` za operacije),
- izboljšana replikacija in
- izboljšana varnost.

V MariaDB 10.0 pa so novi [25]:

- shranjevalni mehanizmi Cassandra, CONNECT, Spider, Sequence,
- ukaza `SHOW EXPLAIN` in `DELETE ... RETURNING`,
- vzporedno podvojevanje in podvojevanje iz več virov,
- knjižnica za regularne izraze PCRE z novimi funkcijami in
- vloge.

V primerjavi z MySQL ponuja MariaDB precej več shranjevalnih mehanizmov. Shranjevalni mehanizem (angl. *storage engine*) je osnovna komponenta SUPB, ki določa, kako so podatki shranjeni, tj. organizirani na disku ali v pomnilniku. Razlikujejo se v načinih indeksiranja, podatkovnih strukturah, nivojih zaklepanja, prisotnosti podpore transakcij in drugih značilnostih.

V tabeli 3.1 so za vse štiri sisteme, ki jih obravnavamo, prikazani shranjevalni mehanizmi, ki jih vključujejo.

3.3 Orodja in odjemalci

Ker je MariaDB *drop-in* zamenjava za MySQL, delujejo z njo načeloma vsi programi, odjemalci, orodja in sistemi, ki delujejo z MySQL. To velja tako za sistemska orodja, kot so `mysql_ugprade`, `mysqlcheck`, `mysql_install_db` in `mysqldump`, kot tudi za grafične odjemalce HeidiSQL, Database Workbench in phpMyAdmin

ter priljubljene sisteme in ogrodja Wordpress, Moodle, Drupal, Plone, Zend Framework idr. [27].

Tabela 3.1: Shranjevalni mehanizmi v SUPB MariaDB in MySQL

	MariaDB 5.5	MariaDB 10.0	MySQL 5.5	MySQL 5.6
MyISAM	✓	✓	✓	✓
MERGE	✓	✓	✓	✓
MEMORY	✓	✓	✓	✓
CSV	✓	✓	✓	✓
Archive	✓	✓	✓	✓
Blackhole	✓	✓	✓	✓
Aria	✓	✓		
InnoDB	✓	✓	✓*	✓*
XtraDB	✓*	✓*		
TokuDB	✓	✓		
Federated	✓**	✓**	✓	✓
Cassandra		✓		
CONNECT		✓		
Spider		✓		
SphinxSE	✓	✓		
OQGRAPH	***	✓		

3.4 Izboljšave v hitrosti

Eden večjih razdelkov na seznamu zmožnosti oziroma prednosti sistema MariaDB so izboljšave v hitrosti. Po besedah razvijalcev so zaradi izpopolnjenega optimizatorja poizvedb hitrejše (in "končno uporabne") gnezdene poizvedbe, optimizirani so tudi stiki (angl. *join*), hitrejši sta izračun kontrolne vsote (angl. *checksum*)

*privzet

**_drop-in_ FederatedX

***onemogočen v tej različici, na voljo v 5.1, 5.2 in 5.3

in pretvorba nabora znakov tabel. Z uporabo shranjevalnega mehanizma Aria za notranje začasne tabele naj bi dosegli tudi večjo hitrost kompleksnih poizvedb [25].

Zadnja izboljšava, ki jo omenjamo v sklopu tega podpoglavja, je segmentiran predpomnilnik ključev (angl. *segmented key cache*). V osnovi MyISAM uporablja predpomnilnik ključev, v katerem hrani bloke tabele, do katerih se je največkrat dostopalo [28]. Ker se ob dostopu do predpomnilnika ključev ta zaklene, se ob hkratnem dostopu velikega števila niti pojavi težava: niti čakajo na sprostitev dostopa. Segmentiran predpomnilnik ključev je razdeljen na segmente (največ 64) in ob dostopu se zaklene le segment, ki ga nit potrebuje, kar močno zmanjša čakanje, saj je verjetnost, da bodo niti potrebovale isti segment, manjša [29].

Naštete obljubljene izboljšave preverimo s testi v drugem delu diplomske naloge; teste opišemo v poglavju 4, rezultate pa predstavimo v poglavju 5.

3.5 MyISAM in Aria

MyISAM je netransakcijski shranjevalni mehanizem, ki je bil privzet shranjevalni mehanizem v MySQL do različice 5.5. Zaradi strukture indeksov je hiter pri branju podatkov in tako najbolj primeren za tabele, v katere se piše malo ali nič. Aria, prej imenovana Maria, je njegova izpeljanka ter funkcionalna in performančna izboljšava, ki v primeru sesutja sistema zagotavlja obnovitev v stanje pred začetkom izvajanega stavka ali prejšnjega zaklepanja tabel (je *crash-safe*) [30]. Na voljo je v sistemu MariaDB od 5.1 naprej, MyISAM pa ekipa iz MariaDB še vedno razvija in dopolnjuje; najnovejša pridobitev je že omenjeni segmentirani predpomnilnik ključev (gl. poglavje 3.4).

MyISAM in Aria ne podpirata tujih ključev in transakcij, podpirata pa polnotekstovne (angl. *fulltext*) indekse, Aria pa tudi (od različice MariaDB 5.2 naprej) navidezne stolpce, ki jih predstavimo kasneje.

V sistemu MySQL lahko za iskanje besed ali izrazov v tabeli uporabimo določilo `LIKE` ali regularne izraze. Slabost teh načinov je, da se mora pri iskanju pregledati celotna tabela in vrednosti primerjati z iskanim izrazom. Pri velikih tabelah so takšne poizvedbe lahko zelo počasne. Tako regularni izrazi kot izrazi v `LIKE` ne omogočajo veliko fleksibilnosti oziroma lahko hitro postanejo prezapleteni. V polnotekstovnem iskanju teh pomanjkljivosti ni, saj temeljijo na polnotekstovnih

indeksih, s katerimi MySQL na osnovi naprednih algoritmov hitro najde vrstice, ki se ujemajo z iskanim izrazom [31].

Tabele so tako v MyISAM kot v Arii v enem od treh formatov; `FIXED` in `DYNAMIC` sta skupna obema, medtem ko je `COMPRESSED` lasten MyISAM in je posledica stiskanja tabel z orodjem `myisampack`, format `PAGE` pa je privzet format v Arii (in edini možen za njene transakcijske tabele). Format tabele določa njeno dolžino (velikost). V tabelah z nespremenljivim (angl. *fixed*), statičnim formatom so vsi stolpci v tabeli enako dolgi, posledično pa tudi vsi zapisi (vrstice). Takšne tabele so hitrejše, saj je zaradi fiksne dolžine znano, kje se začne posamezna vrstica, vendar pa zasedajo več prostora. Dinamični format je nekoliko kompleksnejši: vsaka vrstica vsebuje dodatne podatke o njeni dolžini in praznih/ničelnih vrednostih v njenih stolpcih, ki se na disk ne shranijo. Takšna tabela zato zavzema samo toliko prostora, kot ga potrebuje. Tabele, ki vsebujejo stolpce spremenljive dolžine (npr. `VARCHAR` ali `BLOB`), so avtomatsko shranjene z dinamičnim formatom, medtem ko je za tabele brez takšnih stolpcev privzet statični format [32] [33].

Aria je ključen del sistema MariaDB, saj se uporablja za notranje začasne tabele, ki se kreirajo med obdelavo poizvedb, npr. pri operacijah `GROUP BY` in `DISTINCT` [34]. Cilj razvijalcev je, da postane Aria polno transakcijski in z MVCC skladen shranjevalni mehanizem, ki bo funkcionalno dosegel InnoDB. Transakcije so deloma že omogočene, in sicer z zapisovanjem sprememb v dnevniško datoteko, ki se uveljavijo ob koncu stavka – tako deluje tudi obnovitev po zrušitvi sistema. Da tabeli zagotovimo "odpornost" proti sesutju, ob kreiranju dodamo določilo `TRANSACTIONAL=1` (tako v 5.5 kot v 10.0 je to sicer privzeta vrednost).

Za zagotovitev polne podpore transakcij mora shranjevalni mehanizem zagotavljati lastnosti ACID, tj. [35] [36]

- atomarnost (angl. *atomicity*): transakcija se izvede v celoti ali pa sploh ne; če pri eni od operacij pride do napake, se razveljavi celotna transakcija,
- konsistentnost (angl. *consistency*): transakcija upošteva pravila, omejitve in prožilce, ki so določeni za podatkovno bazo oziroma tabelo; če zapisani podatki povzročijo neveljavno stanje, se transakcija razveljavi,
- izolacijo (angl. *isolation*): dokler se transakcija ne izvede do konca, so rezultati njene izvedbe skriti,
- trajnost (angl. *durability*): rezultat izvedene (potrjene) transakcije je trajno

shranjen in dostopen tudi v primeru napak, sesutij in prekinitev napajanja.

Druga pomembna komponenta, ki smo jo omenili, je MVCC, *multi-version concurrency control*) ali nadzor nad sočasno uporabo z več različicami. Gre za mehanizem, ki omogoča več odjemalcem (nitim) hkraten dostop do istega podatka – tudi, če se ta v istem trenutku posodablja. Pri preprostejši metodi zagotavljanja konsistentnosti podatkov in nadzora nad sočasno uporabo se podatek (vrstica) za druge niti zaklene, dokler ni zahteva ene dokončana; niti tako čakajo na sprostitev dostopa. Pri MVCC se v SUPB hrani več različic podatka – posnetkov, ki se ustvarijo ob njegovem kreiranju in spremembah. Odjemalec, ki želi nek podatek prebrati, dobi zadnjo različico, medtem pa lahko drug odjemalec isti podatek posodablja – vendar do konca zahteve sprememba ne bo vidna prvemu (in ostalim). S spremembo podatka se ustvari njegova nova različica, ki je sedaj najnovejša in bo na voljo nadaljnjim zahtevam [37] [38].

Zmožnosti, ki bodo v Arii implementirane v prihodnjih verzijah, so razveljavitev transakcije (*rollback*), zaklepanje vrstic (namesto celotne tabele) ter sočasnost posodobitev in izbrisov. V dokumentaciji nismo zasledili podatka o tem, ali je v načrtu tudi podpora za tuje ključe, s katerimi bi se Aria funkcionalno približala InnoDB.

3.6 InnoDB in XtraDB

InnoDB je transakcijski shranjevalni mehanizem za MySQL, privzet v verzijah od 5.5 dalje. Podpira transakcije in tuje ključe ter je skladen z ACID, zaradi česar je v primerjavi z MyISAM bolj v koraku s potrebami aplikacij, ki uporabljajo MySQL, in je bil zato logična izbira za nov shranjevalni mehanizem.

XtraDB je Perconina izboljšava InnoDB, ki obljublja boljše performanse, skalabilnost in izkoriščenost pomnilnika. Vključuje InnoDB-jevi z ACID skladno zasnovo in napredno arhitekturo MVCC ter dodaja nove funkcionalnosti, zasnovane na podlagi zahtev uporabnikov in izkušenj podjetja (razširjen prikaz stanja InnoDB, štetje mrtvih zank, hitra kontrolna vsota InnoDB idr.) [39] [40]. V MariaDB je vključen in privzet od različice 5.1.

Od MySQL 5.6 in MariaDB 10.0 InnoDB oz. XtraDB podpira tudi polnotekstovne indekse, s čimer odpade še zadnji razlog za uporabo MyISAM.

3.7 TokuDB

TokuDB, produkt podjetja Tokutek, je visokoperformančni shranjevalni mehanizem za MySQL in MariaDB, ki na osnovi tehnologije indeksiranja s fraktalnimi drevesi močno izboljšuje performanse ter povečuje skalabilnost in kompresijo podatkov. V primerjavi z InnoDB zagotavlja (vsaj) 2,5- do 5-krat večjo kompresijo in 100-krat hitrejše vstavljanje, poleg tega pa omogoča vroče indeksiranje ter dodajanje, preimenovanje in brisanje stolpcev (angl. *hot indexing*, *hot column addition/rename/deletion*) z zelo majhnim časom izpada [41]. V običajnem scenariju se med izvedbo teh operacij tabela zaklene in tako za poizvedbe ni na voljo, dokler se operacija ne zaključi – kar lahko traja. TokuDB z vročim indeksiranjem (dodajanjem, preimenovanjem, brisanjem) to omejitev zaobide in pri takšni operaciji je čas izpada delovanja le tolikšen, kot ga potrebuje MySQL/MariaDB, da po spremembi zapre in ponovno odpre tabelo [42].

V primeru iz zapisa na spletni strani Tokutek je InnoDB za ustvarjanje indeksa potreboval dobrih 31 minut, tabela pa je bila ves ta čas zaklenjena, medtem ko je v TokuDB takšna operacija trajala dobrih 9 minut, tabela pa je bila zaklenjena le 2 sekundi, in to po koncu operacije [42].

V drugem primeru, ko so v tabelo z dobro milijardo vrstic dodali stolpec, je bila razlika v časih izvedbe še večja: v InnoDB se je operacija izvajala skoraj 18 ur, v TokuDB pa nekaj več kot 3 sekunde [43]. S toliko krajšim časom, potrebnim za izvedbo operacije, oziroma minimalnim izpadom delovanja med izvedbo je TokuDB primeren tudi – ali pa predvsem za – zelo velike tabele (velike podatke, angl. *big data*) in/ali tabele, ki jih uporabljajo produkcijske aplikacije.

TokuDB je transakcijski, skladen z ACID (torej transakcijski) in MVCC ter zasnovan kot vtičnik in zamenjava za, kot navajajo na spletni strani podjetja, katerikoli shranjevalni mehanizem [44] (čeprav to ne drži, saj se od npr. InnoDB razlikuje že v omejitvah tujega ključa). V MariaDB je na voljo v samem sistemu, omogočimo ga z ukazom v ukazni vrstici `mysql` ali z zapisom v ustrezno konfiguracijsko datoteko [45]. Performančne teste smo zato izvedli v sistemu MariaDB, v poglavju z rezultati pa te primerjamo s performansami XtraDB (v MariaDB) in InnoDB (v MySQL).

3.8 Navidezni stolpci

Navidezni stolpci (angl. *virtual columns*) so stolpci, katerih vrednost se izračuna samodejno na osnovi določenega izraza (funkcije ali kombinacije funkcij), običajno iz vrednosti drugih stolpcev tabele. Navidezni stolpci niso del SQL-standarda, vendar pa jih poznajo in implementirajo priljubljeni SUPB Microsoft SQL Server, Oracle, Firebird in MariaDB. V slednjem so na voljo v različicah od 5.2 naprej.

Uporabljajo se lahko v tabelah s shranjevalnim mehanizmom, ki jih podpira: InnoDB, XtraDB, MyISAM, Aria in CONNECT [46]. V uradni izdaji MySQL takšne funkcionalnosti še ni, so pa generirani stolpci (angl. *generated columns*) na voljo v raziskovalni (*labs*) različici 5.7.5. Pri obeh sistemih gre za delo istega razvijalca, Andreya Zhakova [47], ki je navidezne stolpce sicer razvil že leta 2008 za MySQL 6.0 [48]. Implementacija je v obeh sistemih tako zelo podobna, kot posledica prilagoditve posameznemu sistemu pa se razlikujejo nekateri izrazi, določila in omejitve.

Navidezni/generirani stolpec je lahko navidezen (angl. *virtual*) – to je tudi privzet tip – ali trajen (angl. *persistent*) oziroma, v MySQL, shranjen (angl. *stored*). Slednji se izračunajo ob vstavljanju ali spreminjanju vrstice in so shranjeni v tabeli ter tako obravnavani kot običajni stolpci, prvi pa se generirajo ob vsaki poizvedbi posebej, če so vanjo vključeni. Prednost enih je torej v tem, da ne zasedajo prostora na disku, drugih pa, da jih ni treba izračunati vsakič znova.

Navidezni stolpec lahko v tabelo dodamo ob kreiranju tabele ali tako, da jo spremenimo, torej s stavkom `CREATE TABLE` ali `ALTER TABLE`. Sintaksa zanj je

```
<type> [GENERATED ALWAYS] AS (<expression>)
  [VIRTUAL | PERSISTENT] [UNIQUE [KEY]] [COMMENT <text>]
```

v MariaDB oziroma

```
<type> [GENERATED ALWAYS] AS (<expression>)
  [ VIRTUAL | STORED ] [UNIQUE [KEY]] [[PRIMARY] KEY]
  [NOT NULL] [COMMENT <text>]
```

v MySQL.

Večina določil je znanih iz sintakse za običajne stolpce, `expression` pa določa zgoraj omenjeni izraz – tistega, ki določa vrednost in je za navidezni stolpec pravzaprav bistven. V MariaDB veljajo naslednja pravila in omejitve:

- izraz biti mora deterministično (ob enakih vhodnih vrednostih vračati isti rezultat),
- izraz ne sme biti oziroma vračati konstantne vrednosti,
- dolžina izraza ne sme presegati 252 znakov,
- izraz ne sme vključevati drugih navideznih stolpcev,
- izraz ne sme uporabljati uporabniško definiranih funkcij, procedur in gnezdenih poizvedb (oziroma česarkoli, kar se nanaša na podatke izven vrstice),
- na navideznem stolpcu tipa `PERSISTENT` lahko ustvarimo indeks, vendar to ne sme biti primarni ključ, lahko pa je tak stolpec del tujega ključa,
- vrednosti navideznega stolpca ne moremo prepisati s svojo vrednostjo; ob vstavljanju vrednosti v tabelo zanj uporabimo `NULL` ali `DEFAULT`.

V MySQL so omejitve (zaenkrat) takšne:

- izraz lahko uporablja vgrajene deterministične funkcije, ne sme pa uporabniško definiranih funkcij in procedur,
- dolžina izraza je omejena z velikostjo datoteke .frm, tj. izraz ne sme biti daljši od 65.536 znakov,
- shranjeni generirani stolpec je lahko del indeksa,
- izraz ne sme vključevati lastnega generiranega stolpca ali drugih, ki so definirani za njim, lahko pa vključuje tiste, ki so definirani prej.

Primer: kreiranje navideznih stolpcev

Poglejmo si primer kreiranja tabele z dvema navideznima stolpcema. Eden od njiju ustvari/določi študentovo vpisno številko na osnovi leta in zaporedne številke vpisa (`id`), drugi pa uporabniško ime, sestavljeno iz prve črke imena in petih črk priimka, ki sta shranjena v "navadnih" stolpcih.

```
CREATE TABLE student (
  id SERIAL PRIMARY KEY,
  priimek VARCHAR(64),
  ime VARCHAR(64),
  leto_vpisa INT(4),
  vpisna_st CHAR(8) AS
    (CONCAT("63", RIGHT(leto_vpisa, 2), LPAD(id, 4, '0'))) VIRTUAL,
```

```
  username VARCHAR(6) AS
    (LOWER(CONCAT(LEFT(ime,1), (LEFT(priimek,5))))) PERSISTENT);
```

Ko v zgoraj ustvarjeno tabelo vstavimo vrstico, izvedeni stavek izgleda takole:

```
INSERT INTO ‘testnabaza‘.‘student‘
  (‘id‘, ‘priimek‘, ‘ime‘, ‘letovpisa‘, ‘vpisna_st‘, ‘username‘)
  VALUES
  (NULL, ’Cetinski’, ’Teja’, ’2007’, NULL, NULL);
```

Na mesto vrednosti navideznih stolpcev se torej vstavita `NULL`. Druga možnost je `DEFAULT`. V primeru, da ti vrednosti prepišemo s svojima, npr.

```
INSERT INTO ‘testnabaza‘.‘student‘
  (‘id‘, ‘priimek‘, ‘ime‘, ‘letovpisa‘, ‘vpisna_st‘, ‘username‘)
  VALUES
  (NULL, ’Cetinski’, ’Teja’, ’2007’, ’63070049’, ’tejace’);
```

bosta ignorirani in namesto njiju vstavljeni izračunani, sistem pa bo za vsako javil opozorilo ali napako (`#1906 The value specified for computed column ’...’ in table ’student’ ignored`).

Dejanske vrednosti, ki se shranijo v tabelo, so `(1, ’Cetinski’, ’Teja’, 2007, ’63070001’, ’tceti’)`.

Tako navidezne kot trajne stolpce lahko spremenimo s stavkom `ALTER TABLE`.

3.9 Dinamični stolpci

Dinamični stolpci omogočajo, kot pove ime, shranjevanje dinamičnih podatkov v tabele; v vsaki vrstici imamo lahko drug(ačen) set stolpcev in s tem podatkov. Uporablja se jih, kadar ne moremo uporabiti klasičnega relacijskega pristopa, tj. tabele z definiranimi stolpci, katerih število se ne spreminja: npr. ko shranjujemo vnose z veliko atributi, množica atributov pa vnaprej ni znana ali je zelo velika. V MariaDB so dinamični stolpci na voljo od različice 5.3 naprej, 10.0.1 pa prinaša nekaj izboljšav [49].

Za shranjevanje takšnih atributov uporabimo dva stolpca: stolpec, ki služi kot primarni ključ (npr. zaporedna številka ali enolično ime), in stolpec tipa `blob`

(zbirka binarnih podatkov, angl. *binary large object*), v katerem bodo shranjeni dinamični stolpci. Tabela lahko seveda vsebuje tudi druge (običajne) stolpce. Ob vstavljanju vrstice nato definiramo stolpce in vrednosti v njih, ki jih lahko kasneje beremo, spremenimo ali dopolnimo z novimi. Za vse te operacije so na voljo posebne funkcije, na kratko opisane v tabeli 3.2, nekatere od njih pa tudi predstavljene v okviru primera v nadaljevanju. Te funkcije so običajne funkcije mysql, kot npr. `SUBSTRING` ali `RAND`, in jih tako uporabljamo v okviru stavkov `SELECT`, `INSERT INTO` in `UPDATE` (gl. primer).

Podatkovne tipe vrednosti, ki jih hranimo v dinamičnih stolpcih, bo ugotovil in določil sistem, lahko pa jih eksplicitno navedemo sami. To je potrebno storiti, če shranjujemo podatek v tipu, ki iz vrednosti same ni razviden, npr. niz `'2015-01-20'` kot datum, tip `DATE` (privzeto bi se shranil kot niz znakov, `CHAR`).

Število dinamičnih stolpcev v okviru "navadnega" stolpca je omejeno s 65.535, skupna velikost takšnega *bloba* pa z vrednostjo strežniške sistemske spremenljivke `max_allowed_packet` (privzeto 1 GB).

Dinamičnih stolpcev zaenkrat še ni mogoče indeksirati, bodo pa razvijalci to funkcionalnost dodali v prihodnosti v primeru, da se bo izkazala potreba (oziroma interes). Začasna rešitev je uporaba navideznega stolpca, na katerem se ustvari indeks [50]; predstavimo jo v primeru v nadaljevanju.

Primer: shranjevanje, spreminjanje in dostopanje do podatkov v dinamičnih stolpcih

Imamo kup predmetov, ki jih želimo podariti ali prodati na bolšjem sejmu. Ker gre za različne stvari, še ne vemo, katere njihove lastnosti bodo bistvene in jih bomo zabeležili. Ustvarimo tabelo, v katero jih bomo vnesli:

```
CREATE TABLE predmeti (
  id SERIAL PRIMARY KEY,
  predmet BLOB);
```

Vstavimo sedaj podatke o treh predmetih z naslednjimi atributi:

- svetilka – višina, premer, leto izdelave, cena;
- žepna ura – znamka, država izvora, opomba;
- škatla – material, barva, dimenzije.

```
INSERT INTO predmeti (predmet) VALUES
  (COLUMN_CREATE('naziv', 'svetilka', 'višina', '140 cm',
    'premer', '45 cm', 'leto', 1921, 'cena', 85)),
  (COLUMN_CREATE('naziv','žepna ura', 'znamka', 'Thiel',
    'izvor', 'Nemčija', 'opomba', 'nedelujoča')),
  (COLUMN_CREATE('naziv','škatla', 'material', 'usnje',
    'barva', 'rjava, pozlačeni detajli',
    'dimenzije', '18 x 12 x 4 cm'));
```

Zdaj lahko s COLUMN_LIST izpišemo vse atribute predmetov:

```
SELECT id, COLUMN_LIST(predmet) FROM predmeti;
+----+-----------------------------------------+
| id | column_list(predmet)                    |
+----+-----------------------------------------+
|  1 | `cena`,`leto`,`naziv`,`premer`,`višina` |
|  2 | `izvor`,`naziv`,`opomba`,`znamka`       |
|  3 | `barva`,`naziv`,`material`,`dimenzije`  |
+----+-----------------------------------------+
```

Izpišimo ime drugega predmeta:

```
SELECT COLUMN_GET(predmet, 'naziv' AS CHAR)
  FROM predmeti WHERE id=2;
+-------------------------------------+
| column_get(predmet, 'naziv' as char) |
+-------------------------------------+
| žepna ura                           |
+-------------------------------------+
```

Dodajmo žepni uri ceno:

```
UPDATE predmeti SET predmet=COLUMN_ADD(predmet, 'cena', 49)
  WHERE id=2;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Še enkrat izpišimo vse vrednosti, tokrat v formatu JSON:

```
SELECT id, COLUMN_JSON(predmet) FROM predmeti;
+----+-------------------------------------------------+
| id | column_json(predmet)                            |
+----+-------------------------------------------------+
|  1 | {"cena":85,"leto":"1921","naziv":"svetilka",    |
|    |  "premer":"45 cm","višina":"140 cm"}            |
|  2 | {"cena":49,"izvor":"Nemčija","naziv":"žepna ura", |
|    |  "opomba":"nedelujoča","znamka":"Thiel"}        |
|  3 | {"barva":"rjava, pozlačeni detajli",            |
|    |  "naziv":"škatla","material":"usnje",           |
|    |  "dimenzije":"18 x 12 x 4 cm"}                  |
+----+-------------------------------------------------+
```

Poiščimo zdaj predmet(e), ki ima(jo) ceno višjo od 50:

```
SELECT id, COLUMN_GET(predmet, 'naziv' AS CHAR) AS naziv,
  COLUMN_GET(predmet, 'cena' AS INT) AS cena FROM predmeti
  WHERE COLUMN_GET(predmet, 'cena' AS INT)>50;
+----+----------+------+
| id | naziv    | cena |
+----+----------+------+
|  1 | svetilka |   85 |
+----+----------+------+
```

Za lažje iskanje po predmetih lahko ustvarimo trajni navidezni stolpec, ki bo služil kot indeks po imenih predmetov:

```
ALTER TABLE predmeti ADD naziv VARCHAR(32) AS
  (COLUMN_GET(predmet, 'naziv' AS CHAR)) PERSISTENT;
CREATE INDEX predmet_naziv ON predmeti(naziv);
```

Tabela 3.2: Funkcije za upravljanje z dinamičnimi stolpci v MariaDB

Funkcija	**Opis**
COLUMN_CREATE	Ustvari stolpec (stolpce). Za vsak stolpec moramo podati ime (v MariaDB 5.3 in 5.5 številko) in vrednost, izbirno tudi podatkovni tip. Funkcija vrne *blob*, ki hrani ustvarjene dinamične stolpce z vrednostmi.
COLUMN_ADD	Doda ali posodobi stolpec (stolpce). Za vsak stolpec moramo podati ime (v MariaDB 5.3 in 5.5 številko) in vrednost, izbirno tudi podatkovni tip. Če stolpec z navedenim imenom že obstaja, bo njegova vrednost prepisana, sicer bo ustvarjen nov. Funkcija vrne posodobljeni *blob*, ki hrani dinamične stolpce.
COLUMN_GET	Vrne vrednost stolpca. Poleg njegovega imena moramo podati tudi podatkovni tip (zahteva SQL-interpretorja).
COLUMN_DELETE	Izbriše stolpec (stolpce). To lahko dosežemo tudi s funkcijo `COLUMN_ADD`, če kot vrednost podamo `NULL`. Funkcija vrne posodobljeni *blob*, ki hrani dinamične stolpce.
COLUMN_EXISTS	Vrne 1, če stolpec s podanim imenom (v MariaDB 5.3 in 5.5 s podano številko) obstaja, sicer vrne 0.
COLUMN_LIST	Vrne seznam z vejicami ločenih imen stolpcev (v MariaDB 5.3 in 5.5 številk stolpcev).
COLUMN_CHECK	(od MariaDB 10.0) Vrne 1, če je podano stolpec veljaven `blob` z dinamičnimi stolpci. Če je, vrne 1, sicer vrne 0.
COLUMN_JSON	(od MariaDB 10.0) Vrne imena in vrednosti stolpcev v formatu JSON ({`"ime":"vrednost","ime2":"vrednost2"...`}).

3.10 Vmesniki NoSQL

3.10.1 InnoDB in Memcached

Novost v MySQL 5.6 je (z)možnost shranjevanja in dostopa do podatkov v načinu NoSQL. Kot smo na kratko omenili že v opisu sistema Percona Server, se pri tem zaobide celotno plast SQL, posledica tega pa je veliko hitrejše shranjevanje podatkov v tabele MySQL in dostopanje do njih (v MySQL 5.7 po enem od testov do 2-krat hitrejše zapisovanje in do 9-krat hitrejše branje v primerjavi s poizvedbami SQL [51]). Do istih podatkov se lahko še vedno dostopa z običajnimi SQL-poizvedbami. Funkcionalnost je na voljo kot vtičnik za InnoDB, in sicer preko programskega vmesnika (*API-ja*) Memcached [52].

Memcached je sistem za porazdeljeno predpomnjenje podatkov, namenjen pohitritvi spletnih aplikacij, tako da je del podatkov iz podatkovne baze shranjen v pomnilniku v obliki ključ-vrednost in se tako izognemo vsakokratnemu poizvedovanju. Njegov programski vmesnik je na voljo za večino priljubljenih programskih jezikov, za MySQL pa je implementiran v obliki vtičnika za prikriti (angl. *daemon*) proces, preko katerega poteka povezava z InnoDB-jevim API-jem.

Za uporabo vtičnika tega najprej namestimo/omogočimo v ukazni vrstici `mysql` z ukazom

```
install plugin daemon_memcached soname "libmemcached.so";
```

in zatem z ukazom

```
source /usr/share/mysql/innodb_memcached_config.sql
```

naložimo tabele vtičnika Memcached za InnoDB, med katerimi je tabela z definiranimi preslikavami med InnoDB in Memcached (pri tem je `/usr/share` mapa, v kateri so datoteke sistema MySQL). Da omogočimo dostop do podatkov neke tabele z Memcached, definiramo vsebovalnik s preslikavami – dodamo ustrezno vrstico v prej omenjeno tabelo `innodb_memcache.containers` [53]. Postopek si bomo ogledali s pomočjo primera, še prej pa naštejmo ukaze za delo z Memcached; ti so:

- `add` za dodajanje,
- `set` za dodajanje ali spreminjanje (glede na to, ali ključ že obstaja),

- `replace`, `append`, `prepend` za spreminjanje,
- `get` za branje in
- `delete` za brisanje vrednosti [54].

Strukturo nekaterih ukazov bomo opisali v sklopu primera v nadaljevanju.

Primer: shranjevanje, spreminjanje in dostopanje do podatkov z Memcached

Kot v primeru z dinamičnimi stolpci (gl. stran 18) ustvarimo tabelo, v katero bomo shranjevali podatke o različnih (starih) predmetih.

```
CREATE TABLE predmeti (id_atribut varchar(255),
  vrednost varchar(255), PRIMARY KEY (id_naziv));
```

V `containers` dodamo vrstico z naslednjimi vrednostmi stolpcev: ime vsebovalnika, ime sheme (baze), ime tabele, stolpec za ključ, stolpec za vrednost; sledijo stolpec z zastavicami, stolpec za vrednosti *cas* in stolpec s časom poteka vrednosti (gre za vrednosti, povezane z Memcachevim predpomnjenjem in obravnavo zahtev) – vse tri nastavimo na 0, kar pomeni, da niso v uporabi; zadnji je stolpec z unikatnim ključem (primarnim ali sekundarnim indeksom).

```
INSERT INTO innodb_memcache.containers VALUES
  ("predmeti", "testnabaza", "predmeti", "id_atribut", "vrednost",
    0, 0, 0, "PRIMARY");
```

Tabela je sedaj pripravljena za NoSQL-dostop; delovanje lahko preizkusimo s pomočjo terminala in protokola `telnet` (privzeta vrata za Memcached so 11211):

```
telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

Dodajmo podatke o prvem predmetu. Uporabili bomo ukaz `add`, ki shrani podatke, če navedeni ključ še nima določene vrednosti. Sintaksa za vse ukaze, ki vrednosti dodajajo ali spreminjajo, je:

```
<command name> <key> <flags> <exptime> <bytes> [noreply]
```

Navesti moramo torej ime ukaza, ključ, pod katerega bomo podatek shranili, zastavice, čas poteka podatka (v sekundah) in velikost podatka v bajtih. Neobvezno lahko dodamo parameter `noreply`, ki strežnik obvesti, naj na zahtevo ne odgovarja. Shranimo sedaj dva podatka o svetilki, ime in ceno. Za (primarni) ključ bomo uporabili kombinacijo zaporedne oziroma identifikacijske številke in naziva atributa (ime, material, cena itd.), ki sta ločena z navpično črto. Za izbiro ustrezne tabele oziroma vsebovalnika, ki smo ga definirali prej, uporabimo njegovo ime (v našem primeru je to "predmeti") in pred njim `@@`, tabelo in ključ pa ločimo s piko.

```
add @@predmeti.1|ime 0 0 8
svetilka
STORED
add @@predmeti.1|cena 0 0 2
85
STORED
```

S pomočjo SQL-poizvedbe lahko zatem preverimo vrednosti v tabeli:

```
mysql> SELECT * from predmeti;
+------------+----------+
| id_atribut | vrednost |
+------------+----------+
| 1|cena     | 85       |
| 1|ime      | svetilka |
+------------+----------+
```

Shranimo še atributa za drugi predmet, usnjeno škatlo, tokrat v ukazni vrstici `mysql`:

```
INSERT INTO predmeti VALUES("2|ime", "škatla"), ("2|material", "usnje");
```

Za branje (izpis) podatka z Memcached uporabimo ukaz `get`, ki mu sledi zahtevani ključ. Izpišimo podatek o materialu drugega predmeta:

```
get @@predmeti.2|material
```

```
VALUE @@predmeti.2|material 0 5
usnje
END
```

Kot vidimo, nam Memcached najprej odgovori s podatki o vrednosti – ključem, ki smo ga zahtevali, zastavicami in dolžino (v bajtih), sledi pa vrstica z dejansko vrednostjo. Prepričali smo se tudi o tem, da InnoDB z Memcached deluje v obe smeri. Obstoječe zapise lahko izbrišemo, jih v celoti zamenjamo ali jim vrednost le pripnemo, lahko jih povečamo ali zmanjšamo. Vsi ukazi in njihovi parametri so na voljo v dokumentaciji projekta (gl. [54]).

3.10.2 HandlerSocket

HandlerSocket je NoSQL-vtičnik za MariaDB (in MySQL). Tako kot vtičnik Memcached omogoča direkten dostop do podatkov, torej mimo plasti SQL, s čimer se izognemo razčlenjevanju stavkov, zaklepanju tabel in nadzoru nad sočasno uporabo, kar bistveno pohitri branje in pisanje podatkov.

HandlerSocket teče in deluje, tako kot vtičnik Memcached za InnoDB, kot prikriti (*daemon*) proces, ki sprejema in izvaja zahteve odjemalcev. Ravno tako ne podpira poizvedb SQL, temveč preproste operacije CRUD [55].

V MariaDB je vtičnik na voljo od verzije 5.3 naprej, potrebno ga je le vklopiti oziroma namestiti, bodisi z zapisom `plugin-load = handlersocket.so` v konfiguracijski datoteki ali z ukazom `install plugin handlersocket soname 'handlersocket.so';` v ukazni vrstici `mysql`. V konfiguracijsko datoteko moramo dodati še naslov in vrata, na katerih HandlerSocket sprejema zahteve – ena samo za branje, druga za branje in pisanje:

```
handlersocket_address="127.0.0.1"
handlersocket_port="9998"
handlersocket_port_wr="9999"
```

Po ponovnem zagonu MariaDB lahko preizkusimo delovanje vtičnika; oglejmo si primer.

Primer: shranjevanje, spreminjanje in dostopanje do podatkov s HandlerSocket

Ustvarimo najprej tabelo za shrambo podatkov o predmetih.

```
CREATE TABLE predmeti_hs (id SERIAL PRIMARY KEY, naziv varchar(32),
  material varchar(50), dimenzije varchar(20), cena int);
```

Kot v primeru z Memcached bomo delovanje preizkusili s pomočjo `telnet`. Za zapisovanje v bazo uporabimo prej definirana vrata za pisanje, tj. `9999`.

```
telnet localhost 9999
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

Za delo s tabelo v podatkovni bazi moramo najprej pripraviti indeks vzpostavljene povezave HandlerSocketa s tabelo. Za vse nadaljnje operacije potrebujemo indeks, da naslovimo pravo podatkovno bazo, tabelo v njej in stolpce, s katerimi operiramo. Sintaksa za kreiranje indeksa je naslednja [55]:

```
P <indexid> <dbname> <tablename> <indexname> <columns> [<fcolumns>]
```

Navesti moramo torej številko indeksa, ime baze, ime tabele, ime ključa v tabeli (v našem primeru je to `id`, uporabimo lahko kar `PRIMARY`) in seznam stolpcev, ločenih z vejico. Zadnja vrednost predstavlja seznam stolpcev za filtriranje in je neobvezna. Vrednosti parametrov so v vseh ukazih ločene s tabulatorjem.

Pripravimo si indeks 0 in izberimo vse stolpce v prej ustvarjeni tabeli.

```
P  0  testnabaza  predmeti_hs  PRIMARY naziv,material,dimenzije,cena
```

HandlerSocket nam ob uspehu odgovori z zaporedjem `0  1`.

V tabelo bi radi shranili podatke – potrebujemo ukaz za vstavljanje. Sintaksa:

```
<indexid> + <vlen> <v1> ... <vn>
```

Potrebujemo indeks (prej ustvarjeni indeks 0), število parametrov (stolpcev) in vrednosti za stolpce, ki jih bomo vstavili. Vstavimo sedaj podatke o svetilki (cena 85 €) in usnjeni škatli velikosti 18 krat 12 krat 4 cm.

```
0    +    4    svetilka            85
0    +    3    škatla    usnje    18 x 12 x 4 cm
```

Stolpce, v katere vrednosti nismo zapisali, smo preskočili, tako da smo se s tabulatorjem samo premaknili na naslednjo vrednost. Pri svetilki sta med nazivom in ceno torej dve tabulatorski mesti. Namesto takšnega pristopa bi lahko ustvarili več indeksov, vsakega s svojim naborom stolpcev; npr. indeks 0 za naziv in ceno, indeks 1 za naziv, material in dimenzije.

Preverimo sedaj podatke v bazi s pomočjo ukazne vrstice `mysql`.

```
MariaDB [testnabaza]> SELECT * from predmeti_hs;
+----+----------+----------+----------------+------+
| id | naziv    | material | dimenzije      | cena |
+----+----------+----------+----------------+------+
|  1 | svetilka |          |                |   85 |
|  2 | škatla   | usnje    | 18 x 12 x 4 cm |    0 |
+----+----------+----------+----------------+------+
```

Izpišimo podatke še s pomočjo HandlerSocketa. Sintaksa za pridobitev (in izpis) podatkov je naslednja:

```
<indexid> <op> <vlen> <v1> ... <vn> [LIM] [IN] [FILTER ...]
```

Indeksu sledijo operator primerjave (`=`, `>`, `<`, `>=` ali `<=`), število stolpcev s ključi, po katerih bomo iskali, in vrednost, ki jo iščemo oziroma s katero primerjamo iskani ključ. Ker HandlerSocket privzeto vrne eno samo vrednost, dodajmo še neobvezni 5. argument, ki določa omejitev števila zapisov.

Izpišimo sedaj največ 20 vrstic, katerih ključ (stolpec `id`) je večji ali enak 1; zahtevi sledi odgovor:

```
0  >= 1  1  20
0  5  1  svetilka      85  2  škatla  usnje  18 x 12 x 4 cm  0
```

Prvo polje odgovora pomeni uspešno izvedeno zahtevo, naslednje ustreza število stolpcev, temu pa sledijo izpisi vrstic – rezultata poizvedbe.

Dodajmo zdaj škatli ceno. Zahteva za spreminjanje/brisanje podatkov je takšna:

```
<indexid> <op> <vlen> <v1> ... <vn> [LIM] [IN] [FILTER ...] MOD
MOD: <mop> <m1> ... <mk>
```

Indeksu sledijo operator (gl. izpis), število stolpcev, ki jih spreminjamo, ključ, število vrstic, ki jih spreminjamo, odmik od začetka tabele (za omejitev vrstic, po katerih iščemo) ter argumenti operacije: vrsta operacije (`U` za posodobitev, `D` za izbris, `+` za povečanje in `-` za zmanjšanje vrednosti) in nove vrednosti stolpcev [56]. V primeru, da vrstico brišemo, nam slednjih ni potrebno navajati.

Posodobimo vrednost stolpca `cena` v vrstici z `id=2`. Ker imamo v HandlerSocket že ustvarjen indeks, ki zajema vse štiri stolpce tabele (tj., razen primarnega ključa), vnesemo vse vrednosti vrstice:

```
0  =  1  2  1  0  U  škatla  usnje  18 x 12 x 4 cm  44
```

Preverimo zapisane podatke:

```
MariaDB [testnabaza]> select * from predmeti_hs;
+----+----------+----------+----------------+------+
| id | naziv    | material | dimenzije      | cena |
+----+----------+----------+----------------+------+
|  1 | svetilka |          |                |   85 |
|  2 | škatla   | usnje    | 18 x 12 x 4 cm |   44 |
+----+----------+----------+----------------+------+
```

S primeroma v tem poglavju smo prikazali osnovni koncept in delovanje vtičnikov za NoSQL. Za oba, Memcached za InnoDB in HandlerSocket, so na voljo knjižnice za pogosto uporabljane programske jezike (C++, Java, PHP, Ruby, Python), s pomočjo katerih shranjevanje in pridobivanje vrednosti izvedemo v okviru programa na enostavnejši in bolj intuitiven način, npr. s funkcijo `get('kljuc')`. Ker

so podatki shranjeni v tabeli znotraj sistema MySQL, ostajajo v veljavi vse omejitve in vsi mehanizmi, torej s pristopom NoSQL ne ”pokvarimo” tabel in podatkov v njih. Še vedno lahko uporabljamo vse zmožnosti, ukaze, funkcije in poizvedbe sistema (My)SQL, kadar potrebujemo direkten dostop do podatka, pa nam je ta na voljo v precej krajšem času in brez režije SQL.

Poglavje 4

Priprava testnega okolja in zmogljivostnih testov

4.1 Sistem

Z namenom omogočiti vzporednost izvajanja testov in s tem kar največjo preglednost nad potekom in rezultati, hkrati pa sistemoma zagotoviti ločene resurse, smo v okviru Arnesove storitve gostovanja virtualnega strežnika "Strežnik po meri" [57] postavili dva navidezna stroja z enako konfiguracijo: 2 procesorskima jedroma in 4 GB pomnilnika. Nanju smo namestili operacijski sistem Ubuntu 14.04 Desktop, na enega od strojev sistem MySQL, na drugega pa MariaDB – oba različice 5.5.40, ki smo ju kasneje nadgradili na 5.6.22 oziroma 10.0.15. Na obeh sistemih sta bila omogočena dostop SSH in oddaljeni dostop do strežnika mysql, ki smo ju potrebovali za izvajanje testov in pri administraciji podatkovnih baz in tabel (v ukazni vrstici in s pomočjo orodja phpMyAdmin).

4.2 Orodje sysbench

Po pregledu razpoložljivih orodij za izvajanje zmogljivostnih testov (angl. *benchmarks*) smo se odločili za sysbench [58], ki poleg testov za merjenje parametrov operacijskega sistema (zmogljivost CPE, pomnilnika, implementacije niti POSIX idr.) vključuje tudi teste zmogljivosti strežnika podatkovnih baz – gre za testni

način OLTP. Glavna razloga za izbiro tega orodja in ne katerega od številnih drugih sta razširjenost (na spletu je o njem mnogo vodičev, člankov in zapisov, tudi na strani MariaDB) ter možnost pisanja in uporabe lastnih skript, ki so od verzije 0.5 naprej napisane v programskem jeziku Lua.

Orodje smo namestili na vsakem od sistemov na navideznem stroju, kjer smo teste tudi izvajali.

Orodje sysbench v okviru testnega načina OLTP ponuja več testov, od enostavnih netransakcijskih, kot sta insert in select, do kompleksnejših, v katerih se izvede več (transakcijskih) poizvedb. Datoteke s testi najdemo v mapi `tests/db`, pri poganjanju pa moramo to pot vključiti kot argument programa. Pred izvajanjem (argument `run`) posameznega testa se najprej pripravi ena ali več testnih tabel, v kateri(h) se bodo vršile operacije, z argumentom `prepare`. Določimo lahko število vrstic (velikost) in shranjevalni mehanizem tabele, za MyISAM pa tudi možnost `MAX_ROWS`, ki jo potrebujemo pri velikih tabelah (nad milijon vrstic).

Med pripravo se izvrši naslednji SQL-stavek:

```
CREATE TABLE ‘sbtest‘ (
  ‘id‘ int(10) unsigned NOT NULL auto_increment,
  ‘k‘ int(10) unsigned NOT NULL default ’0’,
  ‘c‘ char(120) NOT NULL default ’’,
  ‘pad‘ char(60) NOT NULL default ’’,
  PRIMARY KEY  (‘id‘),
  KEY ‘k‘ (‘k‘)
) ENGINE = table_engine;
```

Ustvari se torej tabela s 4 stolpci, po dvema tipa celo število (`int`) oziroma zaporedje znakov (`char`). Prvi stolpec, `id`, je primarni ključ, drugi, `k`, pa ključ ali indeks. Tabela se nato napolni z določenim številom vrstic; če ga ne navedemo, se uporabi privzeta vrednost 10.000.

Primer ukaza za pripravo tabele s shranjevalnim mehanizmom InnoDB in 100.000 vrsticami za test `insert`:

```
./sysbench --test=tests/db/insert.lua --oltp-table-size=100000
 --mysql-table-engine=innodb --mysql-db=testnabaza --mysql-user=root
 --mysql-password=geslo prepare
```

Po zagonu tega ukaza se izpiše obvestilo u ustvarjanju in polnjenju tabele:

```
sysbench 0.5:  multi-threaded system evaluation benchmark

Creating table 'sbtest1'...
Inserting 100000 records into 'sbtest1'
```

Test lahko izvedemo z več nitmi, izvajanje pa lahko omejimo časovno ali s številom operacij (zahtev), ki se bodo izvedle v sklopu testa – ali oboje. Vklopimo lahko tudi možnost sprotnega poročanja, s čimer spremljamo dogajanje in odzivnost. Če sedaj za prej pripravljeno tabelo poženemo test `insert` na 4 nitih za največ 5 sekund ali največ 5000 zahtev s poročanjem vsako sekundo:

```
./sysbench --test=tests/db/insert.lua --num-threads=4 --max-time=5
 --max-request=5000 --oltp-table-size=100000 --mysql-db=testnabaza
 --mysql-user=root --mysql-password=geslo --report-interval=1 run
```

dobimo takle izpis:

```
sysbench 0.5: multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 4
Report intermediate results every 1 second(s)
Random number generator seed is 0 and will be ignored


Threads started!

[   1s] threads: 4, tps: 0.00, reads: 0.00, writes: 927.91, response
     time: 5.95ms (95%), errors: 0.00, reconnects: 0.00
[   2s] threads: 4, tps: 0.00, reads: 0.00, writes: 999.02, response
     time: 5.77ms (95%), errors: 0.00, reconnects: 0.00
[   3s] threads: 4, tps: 0.00, reads: 0.00, writes: 901.01, response
     time: 6.80ms (95%), errors: 0.00, reconnects: 0.00
[   4s] threads: 4, tps: 0.00, reads: 0.00, writes: 608.99, response
     time: 15.06ms (95%), errors: 0.00, reconnects: 0.00
```

```
[  5s] threads: 4, tps: 0.00, reads: 0.00, writes: 905.99, response
    time: 6.58ms (95%), errors: 0.00, reconnects: 0.00
OLTP test statistics:
    queries performed:
        read:                            0
        write:                           4346
        other:                           0
        total:                           4346
    transactions:                        0      (0.00 per sec.)
    read/write requests:                 4346   (868.50 per sec.)
    other operations:                    0      (0.00 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          5.0040s
    total number of events:              4346
    total time taken by event execution: 19.9799s
    response time:
         min:                                  1.49ms
         avg:                                  4.60ms
         max:                                119.82ms
         approx. 95 percentile:                6.79ms

Threads fairness:
    events (avg/stddev):           1086.5000/3.20
    execution time (avg/stddev): 4.9950/0.00
```

Izpisana statistika nam med drugim pove, koliko poizvedb se je izvedlo in kakšne so bile (bralne, pisalne, transakcije ...), minimalni, povprečni in maksimalni odzivni čas posamezne zahteve in skupni čas izvajanja testa. Za nas so najbolj zanimivi podatki o odzivnih časih in številu operacij, ki se izvedejo v eni sekundi. V naslednjem poglavju na osnovi teh podatkov predstavimo rezultate testov za različne operacije na obeh SUPB in z več shranjevalnimi mehanizmi.

Ustvarjeno tabelo lahko po končanem delu izbrišemo s pomočjo orodja samega, in sicer z argumentom (načinom) `cleanup`. Vključimo podatke za povezavo z bazo in test, za katerega je bila tabela pripravljena. Zgoraj ustvarjeno tabelo tako "pospravimo" z naslednjim ukazom:

```
./sysbench --test=tests/db/insert.lua --mysql-db=testnabaza
 --mysql-user=root --mysql-password=geslo cleanup
```

4.3 Vključeni testi

Kot smo že omenili, je v programu samem na voljo več testov. V nadaljevanju so opisani oziroma so navedene poizvedbe, ki jih izvajajo.

insert

Pri tem testu se v tabelo vstavi določeno število vrstic z naključno generiranimi vrednostmi za stolpce razen prvega; ta ima, kot smo videli prej v stavku `CREATE TABLE`, lastnost `AUTO_INCREMENT` in se povečuje samodejno. Stavek, ki se izvede:

```
INSERT INTO sbtest (id, k, c, pad) VALUES (i, N, c_val, pad_val);
```

Vrednost `i` je enaka inkrementu v zanki, `N`, `c_val` in `pad_val` pa se naključno generirajo; prvi zavzame vrednost med 1 in številom, ki ustreza velikosti tabele, druga dva pa sta 119 oz. 59 znakov dolg niz.

select

Izvede se poizvedba, ki vrne vrednost stolpca `pad`:

```
SELECT pad FROM sbtest WHERE id=N;
```

Pri tem je `N` naključno število med 1 in številom, ki ustreza velikosti tabele.

delete

Izbriše se vrstica, ki vsebuje ustrezni primarni ključ:

```
DELETE FROM sbtest WHERE id=N;
```

`N` se generira kot v prejšnjih primerih.

update_index

Vrednost ključa (stolpec `k`) se zamenja z novo, ki je za ena večja od trenutne:

```
UPDATE sbtest SET k=k+1 WHERE id=N;
```

`N` se generira kot v prejšnjih primerih.

update_non_index

Vrednost stolpca `c` se zamenja z novo, ki se generira kot pri kreiranju tabele in vstavljanju vrstice – je 119 znakov dolg niz:

```
UPDATE sbtest SET c=c_val WHERE id=N;
```

`N` se generira kot v prejšnjih primerih.

Naslednja testa uporabljata pripravljeni stavek (angl. *prepared statement)*, torej stavek, v katerega se parametri vstavijo kasneje, tik preden se izvede dejanska poizvedba. Parametrov je privzeto 10 (pri točkovni poizvedbi) oziroma 20 (pri poizvedbi z razponi), lahko pa drugačno število določimo ob zagonu testa.

Vrstice ustvarjene tabele se pri obeh napolnijo z naslednjimi vrednostmi: od 1 do števila, ki ustreza velikosti tabele, za `id` in `k`, `c` ostane prazen, v `pad` pa se vstavi niz "`qqqqqqqqqqwwwwwwwwwweeeeeeeeeerrrrrrrrrrttttttttttt`".

Tabela je glede na število niti razdeljena na ustrezno število segmentov, v okviru katerih se naključno generirajo vrednosti parametrov; tako se segmenti med seboj ne prekrivajo.

Testa sta bila zasnovana s strani ekipe MariaDB za testiranje segmentiranega predpomnilnika ključev za MyISAM. Rezultate izvedenih testov predstavljamo v poglavju 5.2.

select_random_points

Poizvedba vrne vse štiri stolpce vrstic, katerih ključ (`k`) se ujema z eno od vrednosti, navedenih v oklepaju (določilo `IN`).

```
SELECT id, k, c, pad FROM sbtest WHERE k IN (?, ?, ? ... ?);
```

select_random_ranges

Poizvedba vrne število vrstic, katerih ključ (`k`) je v enem od razponov števil, navedenih v oklepaju (določilo `BETWEEN ... AND`).

```
SELECT count(k) FROM sbtest WHERE k BETWEEN ? AND ?
 OR k BETWEEN ? AND ? OR ... OR k BETWEEN ? AND ?;
```

oltp

Test `oltp` je od vseh najkompleksnejši, izvede pa se transakcijsko, če so transakcije v tabeli podprte (tj. če je izbran transakcijski shranjevalni mehanizem, npr. InnoDB), oziroma tabelo pred poizvedbo zaklene in po njej odklene. Vsaka poizvedba se torej začne z `BEGIN` oziroma `LOCK TABLES sbtest WRITE` in konča s `COMMIT` oziroma z `UNLOCK TABLES`.

Test vsebuje več različnih stavkov in operacij, ob zagonu pa lahko določimo njihovo število.

V posamezno transakcijo so vključene naslednje poizvedbe:

```
SELECT c FROM sbtest WHERE id=N;
SELECT c FROM sbtest WHERE id BETWEEN N AND M;
SELECT SUM(K) FROM sbtest WHERE id BETWEEN N and M;
SELECT c FROM sbtest WHERE id between N and M ORDER BY c;
SELECT DISTINCT c FROM sbtest WHERE id BETWEEN N and M ORDER BY c;
UPDATE sbtest SET k=k+1 WHERE id=N;
UPDATE sbtest SET c=c_val WHERE id=N;
DELETE FROM sbtest WHERE id=N;
INSERT INTO sbtest VALUES (i, N, c_val, pad_val);
```

`N` predstavlja začetek razpona in je generiran naključno, kot v prejšnjih primerih. `M` predstavlja konec razpona in je določen kot `oltp_range_size-1`, ta pa je podan kot argument oziroma se uporabi privzeta vrednost 100. `c_val` in `pad_val` se generirata kot v testu `insert` (gl. razdelek insert na strani 35).

Test lahko poženemo v bralnem načinu – v tem primeru se izvede 14 stavkov SELECT, 10 enostavnih ter po eden od preostalih –, ali pa v bralno-pisalnem, ki doda 2 poizvedbi UPDATE, eno DELETE in eno INSERT. Z dodanima stavkoma

za začetek in konec transakcije (oziroma za zaklepanje in odklepanje tabele) je v posamezni transakciji torej 20 poizvedb.

4.4 Prilagoditve v kodi

Za pripravo tabele (kreiranje in polnjenje) se pri večini testov uporabi koda v `common.lua`, testa `select_random_points` in `select_random_ranges` pa imata svoji funkciji `prepare`, v katerih je ustrezen stavek `CREATE TABLE`. V vseh primerih gre za stavek, ki smo ga opisali v poglavju 4.2.

Dobra polovica testov je bila izvedenih na sistemu MariaDB s shranjevalnim mehanizmom Aria, ki je privzeto transakcijski. Ker smo njegovo zmogljivost v osnovi primerjali z zmogljivostjo shranjevalnega mehanizma MyISAM, je bilo potrebno zagotoviti enakost med njima, zato smo na sistemu z MariaDB prej omenjenemu stavku `CREATE TABLE` dodali določilo `TRANSACTIONAL = 0`, ki v tabelah z Ario izklopi transakcijski način.

Za netransakcijsko delo z Ario smo morali prilagoditi tudi tisti del kode v `otlp.lua`, ki poizvedbe obda z ustreznimi stavki za začetek in konec transakcije (oziroma zaklep in odklep tabele). Izvirnemu delu

```
if (((db_driver == "mysql") or (db_driver == "attachsql")) and
  mysql_table_engine == "myisam") then
    begin_query = "LOCK TABLES sbtest WRITE"
    commit_query = "UNLOCK TABLES"
else
    begin_query = "BEGIN"
    commit_query = "COMMIT"
end
```

smo v pogoj dodali še `or mysql_table_engine == "aria"`. Aria se tako v vseh testih obnaša netransakcijsko – kot MyISAM.

4.5 Dodatni testi

Ker osnovni testi, ki so v orodje sysbench že vključeni, ne zadostujejo našim potrebam, tj. ne pokrivajo vseh aspektov in operacij, katerih zmogljivost smo želeli

preveriti in primerjati, smo spisali dodatne teste, ki vključujejo oziroma izvajajo želene operacije in so predstavljeni v nadaljevanju.

checksum

Prva od takšnih operacij je `CHECKSUM TABLE`, ki vrne kontrolno vsoto tabele. Za testiranje je bilo dovolj po vzoru skripte `insert.lua` narediti novo in SQL-stavek za vstavljanje vrstic spremeniti v:

```
CHECKSUM TABLE sbtest;
```

Celotna datoteka je na voljo v dodatku A.

charset_convert

Operacija, katere izvajanje naj bi bilo v MariaDB hitrejše, je tudi pretvorba nabora znakov tabele. Da bi to "obljubo" preverili, smo pripravili skripto z naslednjim SQL-stavkom:

```
ALTER TABLE sbtest CONVERT TO CHARACTER SET 'utf8' COLLATE
 'utf8_unicode_ci';
```

Privzet nabor znakov je sicer `latin1_swedish_ci`.
Celotna datoteka je na voljo v dodatku A.

Za preizkus izboljšanega optimizatorja poizvedb v MariaDB smo pripravili gnezdene poizvedbe (angl. *subqueries*), poizvedbe z združitvijo (angl. *join queries*) in poizvedbe z agregacijsko funkcijo *GROUP BY*. Izpisane so v nadaljevanju. Ker vse poizvedbe potrebujejo več med seboj povezanih tabel, smo jih izvajali na vzorčni podatkovni bazi Sakila [59] in ne na sproti generiranih tabelah. Vse tri datoteke so na voljo v dodatku A.

innerjoin

Poizvedba vrne imena in telefonske številke strank, ki imajo izposojene DVD-je, katerih rok vračila je potekel, in naslove teh DVD-jev [60]:

```
SELECT CONCAT(customer.last_name, ', ', customer.first_name)
  AS customer, address.phone, film.title FROM rental
  INNER JOIN customer ON rental.customer_id = customer.customer_id
  INNER JOIN address ON customer.address_id = address.address_id
  INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
  INNER JOIN film ON inventory.film_id = film.film_id
  WHERE rental.return_date IS NULL
  AND rental_date + INTERVAL film.rental_duration DAY < CURRENT_DATE();
```

groupby

Poizvedba za vsako stranko vrne njeno ime in priimek ter skupno število izposoj.

```
SELECT CONCAT(customer.first_name, ' ', customer.last_name) AS name,
  COUNT(rental.rental_id) AS rentals FROM rental, customer
  WHERE rental.customer_id=customer.customer_id
  GROUP BY customer.customer_id;
```

subquery

Poizvedba vrne imena vseh strank, ki so si kdaj izposodile znanstvenofantastični film (kategorija z `id=14`).

```
SELECT customer_id,
  CONCAT(customer.first_name, ' ', customer.last_name)
  AS name FROM customer
  WHERE customer_id IN (
    SELECT customer_id FROM rental WHERE inventory_id IN (
      SELECT inventory_id FROM inventory WHERE film_id IN (
        SELECT film_id FROM film_category WHERE category_id=14)));
```

fulltext

V podatkovni bazi Sakila so v posebni tabeli, ki uporablja shranjevalni mehanizem MyISAM, v dveh stolpcih shranjeni naslovi in opisi filmov. V tabeli je polnotekstovni indeks, ki zajema ta stolpca.

Za primerjavo polnotekstovnega indeksa v vseh štirih glavnih shranjevalnih mehanizmih smo pognali poizvedbo nad stolpcema z naslovi in opisi filmov, ki sta vključena v indeks. Poizvedba vrne filme, ki imajo v naslovu ali opisu besedo "dinosaur" ali "space".

```
SELECT * FROM film_text WHERE MATCH(title, description)
  AGAINST ('dinosaur space');
```

dynamic_*

Za primerjavo performans poizvedb z običajnimi stolpci in poizvedb z dinamičnimi oziroma navideznimi stolpci smo pripravili več testov. Za osnovo smo vzeli primer iz poglavja o dinamičnih stolpcih (gl. stran 18) in pripravili dve tabeli: eno, ki vsebuje stolpec za vsak atribut, ki ga lahko predmetu določimo, in drugo, ki atribute hrani v *blobu* z dinamičnimi stolpci. Slednja ima poleg tega še 4 navidezne stolpce; dva trajna, eden hrani dinamični stolpec `opis`, drugi dinamični stolpec `cena`, in dva navidezna, od katerih eden pridobi dinamični stolpec `naziv`, drugi pa izračuna ceno v funtih, tako da dinamični stolpec `cena` pomnoži z 0,74.

Shemi tabel sta prikazani na sliki 4.1.

Za testiranje smo nato spisali poizvedbe; s časom izvedbe poizvedovanja po navadnih stolpcih smo želeli primerjati čas, potreben za pridobitev vrednosti iz dinamičnih stolpcev, s časom za pridobitev podatkov in operacijo množenja navadnega stolpca z neko vrednostjo (trenutnega tečaja za britanski funt) pa čas, potreben za pridobitev/izračun navideznega stolpca.

Poizvedbe so torej:

```
SELECT naziv, leto, barva, opis, cena
  FROM dynamic_cela WHERE id=N
```

za "navadno" poizvedbo, tj. poizvedbo v tabeli z običajnimi stolpci,

```
SELECT COLUMN_GET(predmet, 'naziv' as CHAR) as naziv,
  COLUMN_GET(predmet, 'leto' as int) as leto,
  COLUMN_GET(predmet, 'barva' as CHAR) as barva,
  COLUMN_GET(predmet, 'opis' as CHAR) as opis,
  COLUMN_GET(predmet, 'cena' as INT) as cena
  FROM dynamic_din WHERE id=N
```

za poizvedbo po dinamičnih stolpcih;

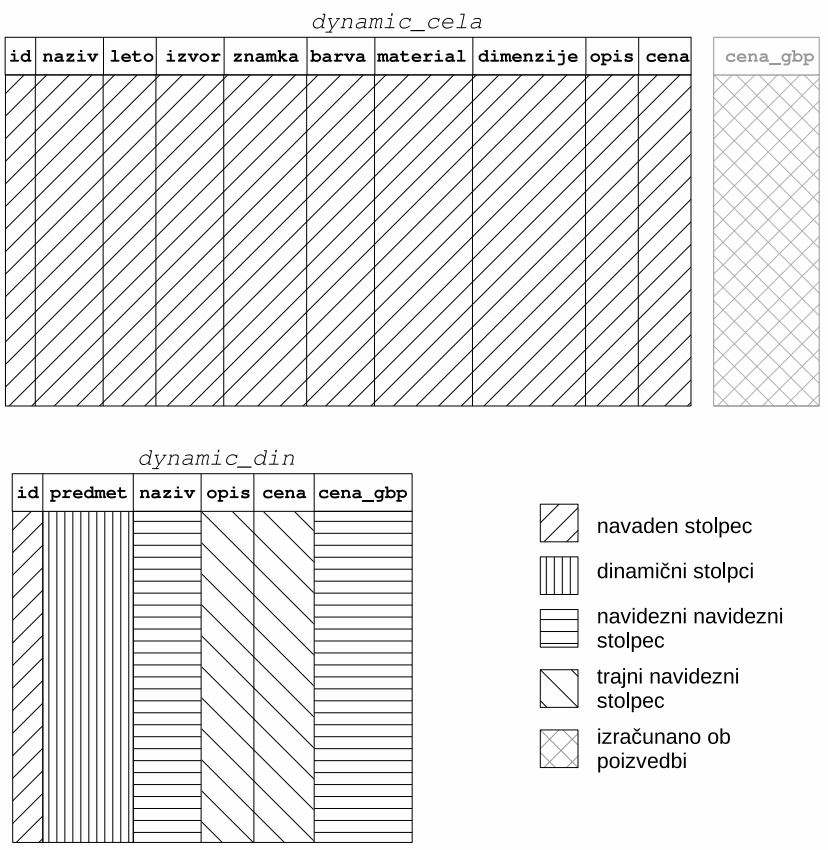
```
SELECT naziv, opis, cena, cena*0.74 as cena_gbp
  FROM dynamic_cela WHERE id=N
```

za poizvedbo v tabeli z navadnimi stolpci in dodatnim izračunom ter

```
SELECT naziv, opis, cena, cena_gbp
  FROM dynamic_din WHERE id=N
```

za poizvedbo po navideznih stolpcih.

Ustrezni skripti, ki kreirata eno in drugo tabelo, in vsi štirje opisani testi so na voljo v dodatku A.



Slika 4.1: Shemi tabel `dynamic_cela` *in* `dynamic_din`

Poglavje 5

Rezultati in analiza

Teste, opisane v poglavju 4, smo pognali na sistemih MariaDB 5.5.40, MySQL 5.5.40, MariaDB 10.0.15 in MySQL 5.6.22. Pri vseh testih smo izvedli po 1000 ponovitev (tj. parameter `--max-requests` je bil enak 1000) z 1, 2, 4, 8 oziroma 16 nitmi, ki predstavljajo vzporedne zahteve (odjemalce), in sicer: pri v sysbench vključenih testih in testih `dynamic` nad tabelami velikosti 100.000, 200.000, 500.000, 1.000.000, 2.000.000 oziroma 5.000.000 vrstic; pri testih `checksum` in `charset_convert` nad tabelami velikosti od 2.500 do 20.000 vrstic s korakom 2.500 ter 100.000 in 200.000 vrstic. Pri testih, izvedenih v podatkovni zbirki Sakila, so tabele različnih velikosti in jih nismo spreminjali.

Izvajanje smo nekoliko avtomatizirali s pripravo bash-skripte, ki ustrezno definira potrebne spremenljivke, izvede ukaze za pripravo tabele, izvedbo testa in brisanje table ter dobljene rezultate iz izpisa shrani v datoteko, ločene z vejicami. Tako so pripravljeni za nadaljnjo obdelavo (prenos v program za preglednice, izris grafov).

Zaradi boljše preglednosti smo rezultate posameznega testa razdelili in jih predstavljamo v sklopu več grafov. Če ni drugače zapisano, so predstavljeni rezultati izvajanja z 1 nitjo.

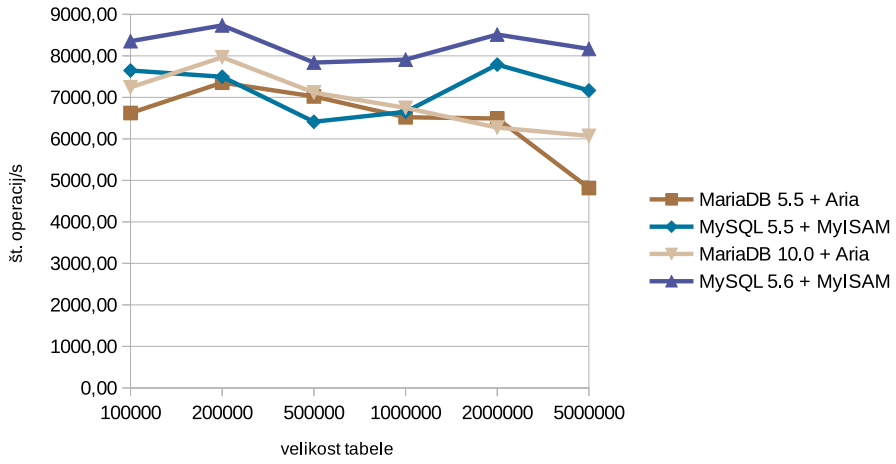
5.1 Osnovne operacije

Poizvedovanje (SELECT)

Na sliki 5.1 so zajeti rezultati testa v tabelah z netransakcijskima shranjevalnima mehanizmoma MyISAM in Aria.

Pri vseh velikostih tabele je najhitrejši MyISAM v novejši različici MySQL; v eni sekundi izvede med 7.839 in 8.734 poizvedb, dohitevata ga starejši MyISAM in Aria iz MariaDB 10.0 s 85–90 % teh vrednosti. Krivulji mehanizma MyISAM v obeh verzijah sistema potekata precej podobno, le na različnih višinah, medtem ko se krivulji Arie deloma prekrivata.

Za oba shranjevalna mehanizma lahko na osnovi teh rezultatov rečemo, da sta bila od prejšnjih različic SUPB performančno izboljšana.

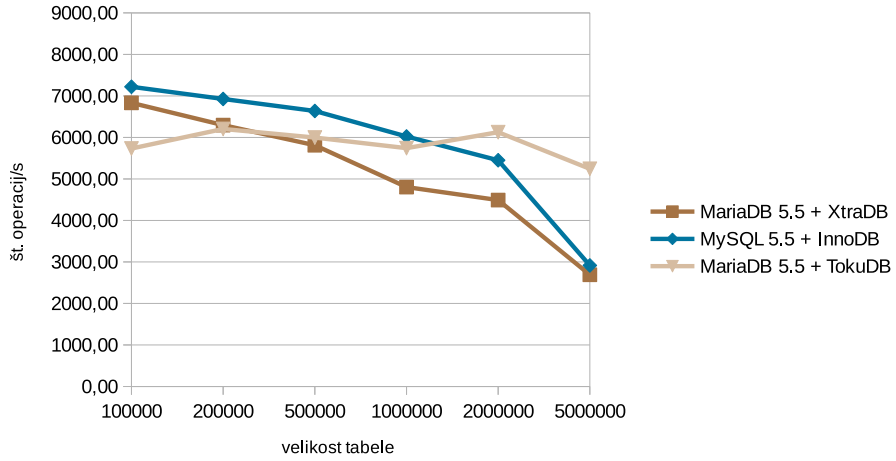


Slika 5.1: Število poizvedb v eni sekundi v netransakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB

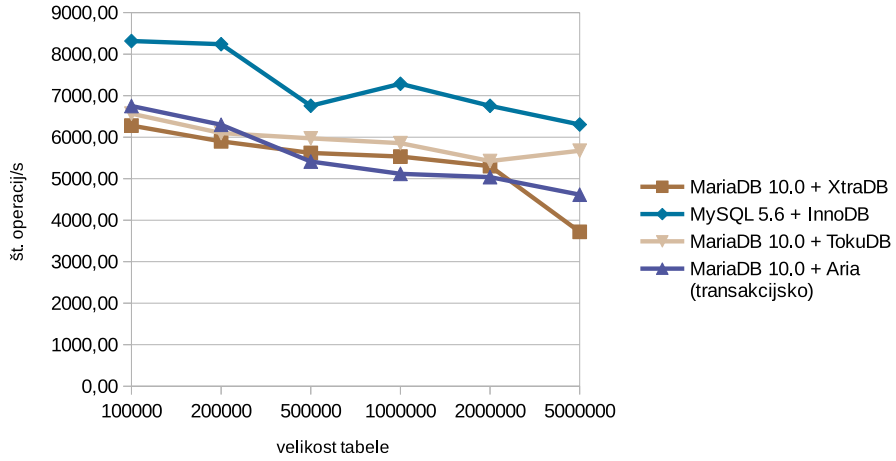
Rezultati poizvedb s transakcijskimi shranjevalnimi mehanizmi InnoDB, XtraDB in TokuDB v sistemih MySQL in MariaDB 5.5.40 so predstavljeni na sliki 5.2, na sliki 5.3 pa so njihovi rezultati v novejših SUPB. V slednjih smo za primerjavo dodali še shranjevalni mehanizem Aria v transakcijskem načinu (določilo `TRANSACTIONAL=1`).

Tudi tu v številu operacij, izvedenih v eni sekundi, vodi MySQL – z InnoDB. Kot pri netransakcijskih shranjevalnih mehanizmih je v novejši različici hitrejši, in

sicer za okrog 15 do 30 %. Vendar pa je TokuDB, ki je v večini točk pod InnoDB, bolj ”umirjen”; vrednosti se gibljejo med 5.244 in 6.126 operacijami/s v starejših sistemih in oziroma med 5.426 in 6.567 v novejših. Predvidevamo, da se TokuDB ne bi vdal niti pri večjih tabelah, medtem ko pri drugih shranjevalnih mehanizmih vidimo upad zmogljivosti pri največji tabeli.



Slika 5.2: Število poizvedb v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB 5.5.40

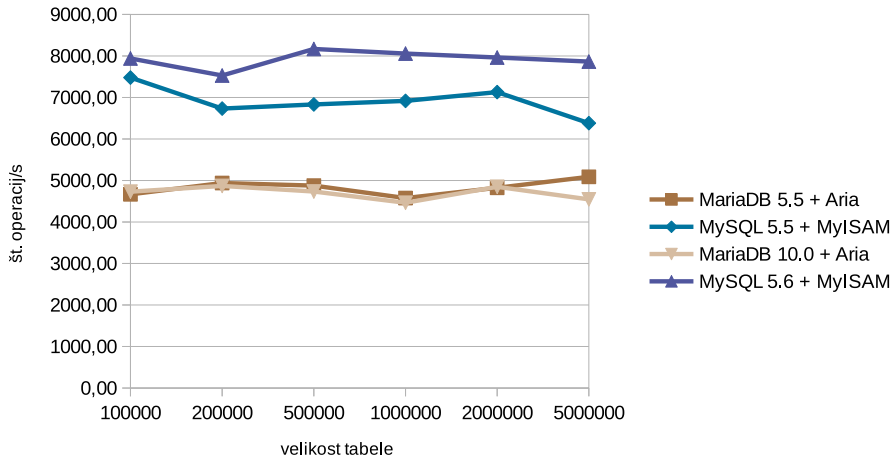


Slika 5.3: Število poizvedb v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL 5.6 in MariaDB 10.0

Vstavljanje vrstic (INSERT INTO)

Rezultati zahtev za vstavljanje vrstic v tabelo z 1 nitjo so kot v prejšnjem razdelku prikazani na treh grafih; za netransakcijska shranjevalna mehanizma na sliki 5.4, za transakcijske pa na slikah 5.5 in 5.6.

Tudi tu vodi MyISAM v novejšem MySQL, in sicer s 7530 do 8169 operacijami na sekundo, sledijo mu MyISAM v MySQL 5.5 (6.382–7.479 operacij/s) ter Aria, katere zmogljivost v novejšem sistemu MariaDB ostaja približno enaka tisti v starejšem (med 4.464 in 5.088 operacij/s).



Slika 5.4: Število vstavljenih vrstic v eni sekundi v netransakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB

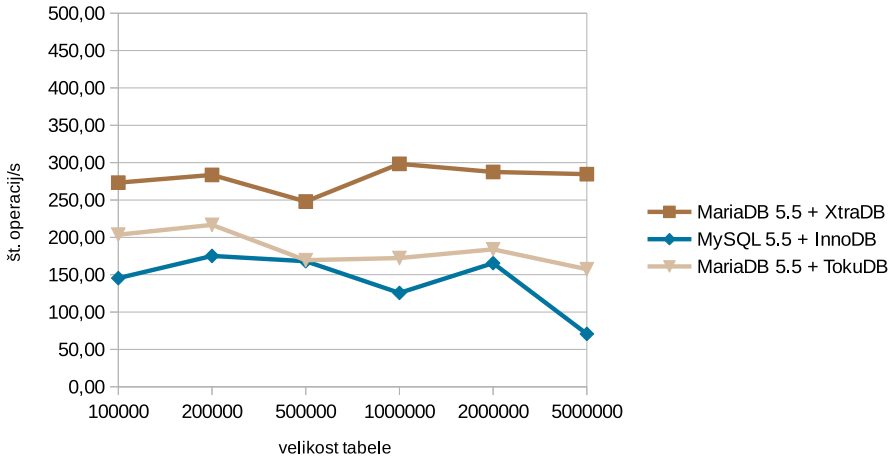
Med transakcijskimi shranjevalnimi mehanizmi se tu bolje izkaže Perconin XtraDB, ki doseže med 248 in 298 operacij/s in je od InnoDB v povprečju hitrejši 1,8-krat, od TokuDB pa 1,4-krat. InnoDB je v MySQL 5.6 hitrejši; izvede med 210 in 238 operacij/s ter prehiti tako TokuDB kot XtraDB v MariaDB 10.0. Presenetljivo je, da je prepustnost slednjih v novejših sistemih slabša kot v starejših.

Vse tri shranjevalne mehanizme sicer močno preseže transakcijska Aria s 388 do 474 operacijami v eni sekundi. Rezultat je navdušujoč, vendar pa se pri več nitih izkaže nezmožnost sočasnosti vstavljanja. Oglejmo si graf s 16 nitmi na sliki 5.7. V primerjavi z ostalimi shranjevalnimi mehanizmi, katerih vrednosti se zadržujejo okrog 1200, Aria ohranja skoraj ravno črto pri 400 operacij/s. Z XtraDB, npr.,

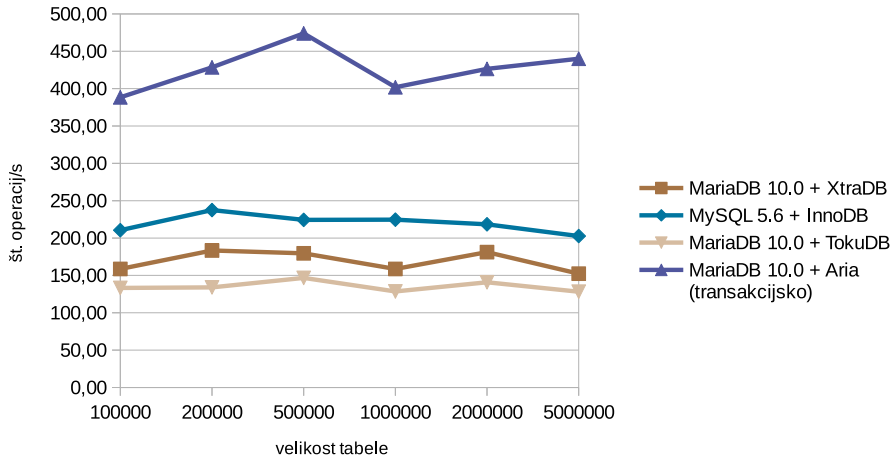
potrebujemo za dosego takšnih vrednostih 4 niti, vendar po drugi strani z uporabo 16 v istem času vstavimo 4-krat toliko vrstic, kot jih zmore Aria.

Po dobljenih rezultatih sodeč, Aria še ne omogoča sočasnih vstavljanj, čeprav njeni razvijalci trdijo drugače. Ob pregledu seznama procesov med izvedbo testa smo ugotovili, da so razen ene niti, ki izvaja stavek `INSERT INTO`, ostale v stanju *Waiting for table level lock* – čakajo torej sprostitev zaklepa tabele, da lahko same pridobijo zaklep.

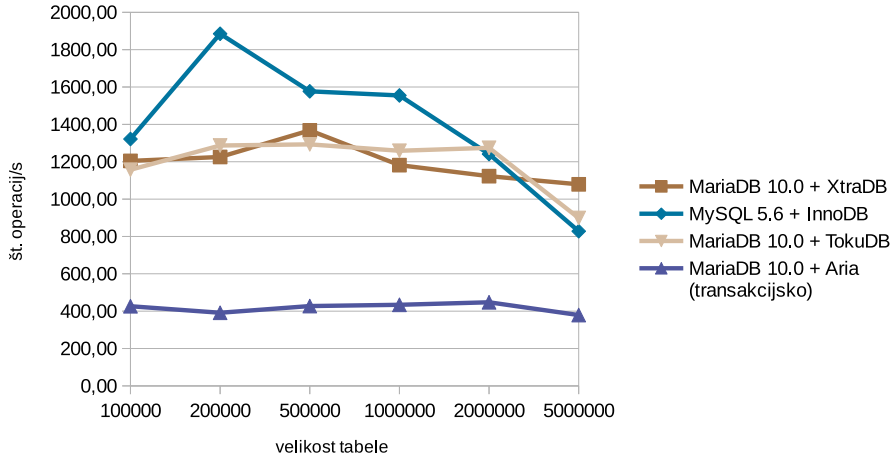
Če upoštevamo to dejstvo oziroma naredimo zaključek na podlagi rezultatov z več nitmi, je torej v prednosti XtraDB, v novejših sistemih pa InnoDB.



Slika 5.5: Število vstavljenih vrstic v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB 5.5.40



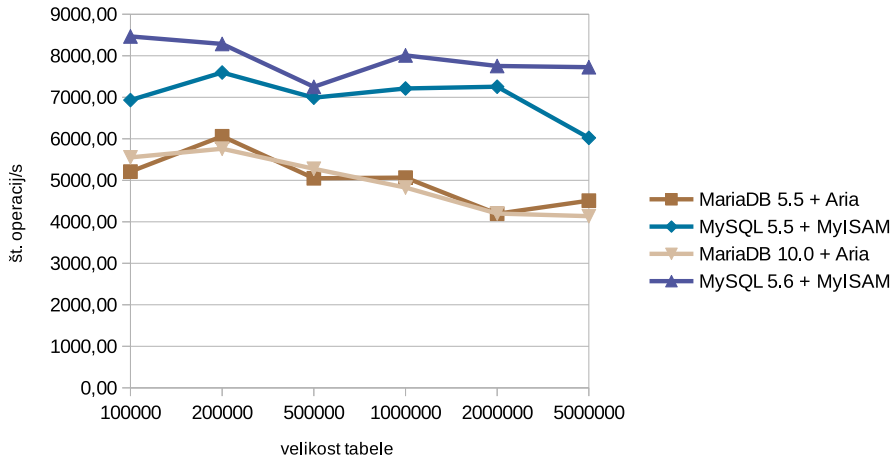
Slika 5.6: Število vstavljenih vrstic v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL 5.6 in MariaDB 10.0



Slika 5.7: Število vstavljenih vrstic v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL 5.6 in MariaDB 10.0 (16 niti)

Brisanje vrstic (DELETE)

Rezultati testa, pri katerem se vrstice brišejo iz tabele, so nekoliko podobni rezultatom vstavljanja; od netransakcijskih ponovno vodi MyISAM v MySQL 5.6 (med 7251 in 8465 operacij/s), ki mu sledi starejši MyISAM, temu pa Aria v obeh sistemih – slika 5.8.

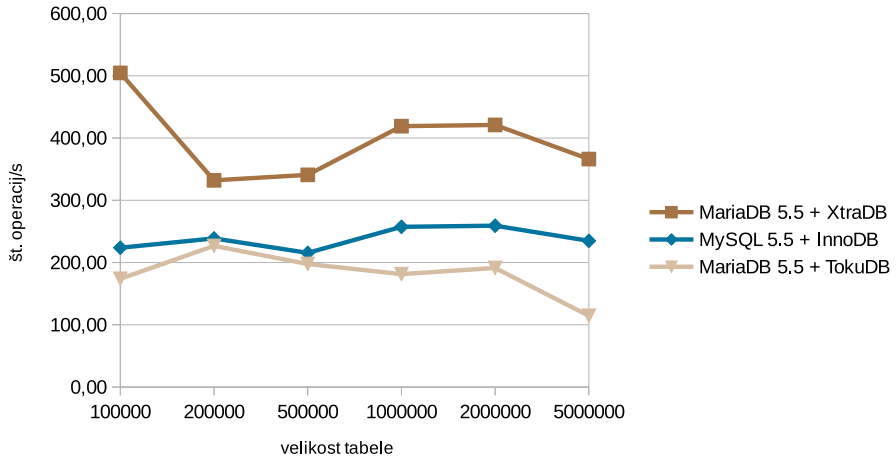


Slika 5.8: Število izbrisanih vrstic v eni sekundi v netransakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB

Od transakcijskih sistemov v starejši verzijah SUPB vodi XtraDB (332–504 operacije/s), za njim je InnoDB z v povprečju 61 % teh vrednosti, še nekoliko manj operacij, med 114 in 227 na sekundo, pa doseže TokuDB. Rezultati so na sliki 5.5.

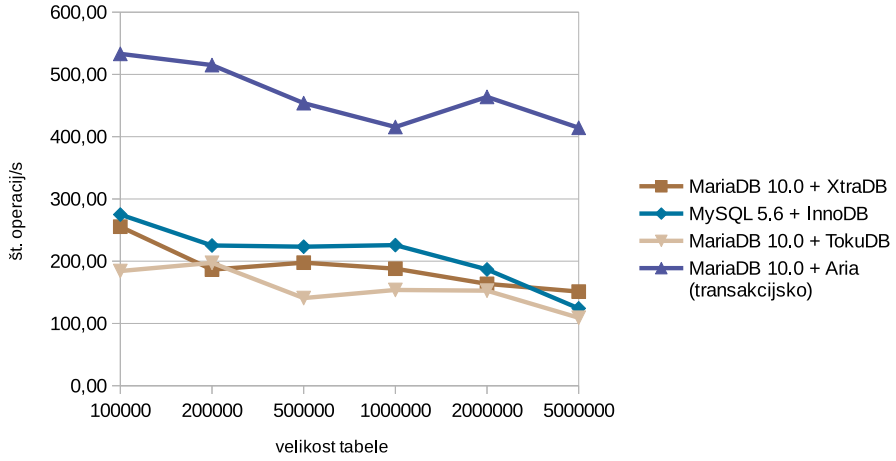
V novejših sistemih se vrstni red spremeni; večinoma vodi InnoDB, katerega krivulji je zelo podobna krivulja XtraDB. Slednji dosega 83–93 % zmogljivosti prvega, ki v sekundi izbriše med 187 in 275 vrstic, pri največji tabeli pa je razmerje obratno: XtraDB izvede 151, InnoDB pa 124 brisanj v sekundi. Ustrezen graf je na sliki 5.10.

Tudi pri tej operaciji je sicer Aria precej hitrejša od ostalih, od 1,8- do 2,7-krat od najhitrejšega, vendar pa pri 8 nitih in več zaradi nezmožnosti sočasnih brisanj močno zaostane. Za ponazoritev smo pripravili stolpčni diagram števila izvedenih izbrisov vrstic iz tabele z milijon vrsticami v vseh štirih shranjevalnih mehanizmih v sistemih MySQL 5.6 in MariaDB 10.0 pri izvedbi z 1, 2, 4, 8 in 16 nitmi. Kot vidimo, višina stolpcev XtraDB, InnoDB in TokuDB narašča s številom



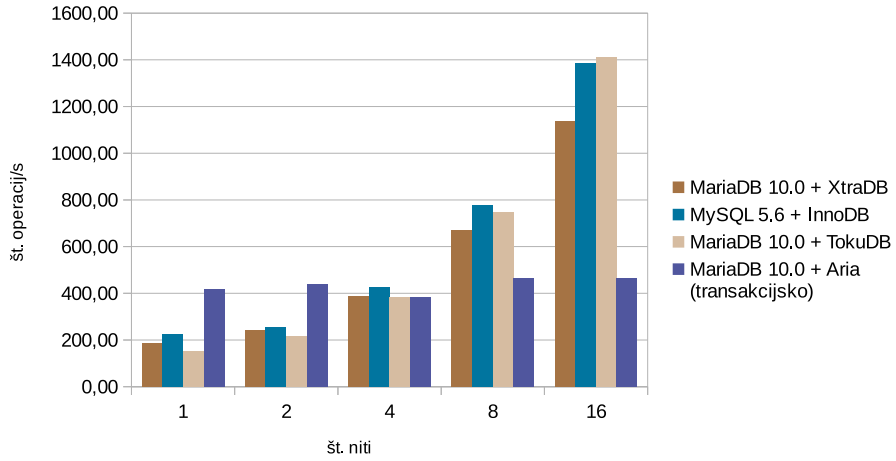
Slika 5.9: Število izbrisanih vrstic v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB 5.5.40

niti, medtem ko se stolpci shranjevalnega mehanizma Aria ves čas gibljejo okrog 400 operacij/s. Pri 4 nitih je tako za malenkost počasnejša, pri 8 in 16 pa dosega le še 60 % oz. 33 % zmogljivosti najhitrejšega shranjevalnega mehanizma.



Slika 5.10: Število izbrisanih vrstic v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL 5.6 in MariaDB 10.0

Kot pri prejšnjem testu, tj. operaciji vstavljanja vrstic, dosega najboljše rezultate MyISAM v MySQL 5.6, od transakcijskih shranjevalnih mehanizmov pa XtraDB v MariaDB 5.5 oziroma, pri manjšem številu niti, Aria.



Slika 5.11: Število izbrisanih vrstic v eni sekundi iz tabele z 1 mio vrstic v transakcijskih shranjevalnih mehanizmih v sistemih MySQL 5.6 in MariaDB 10.0

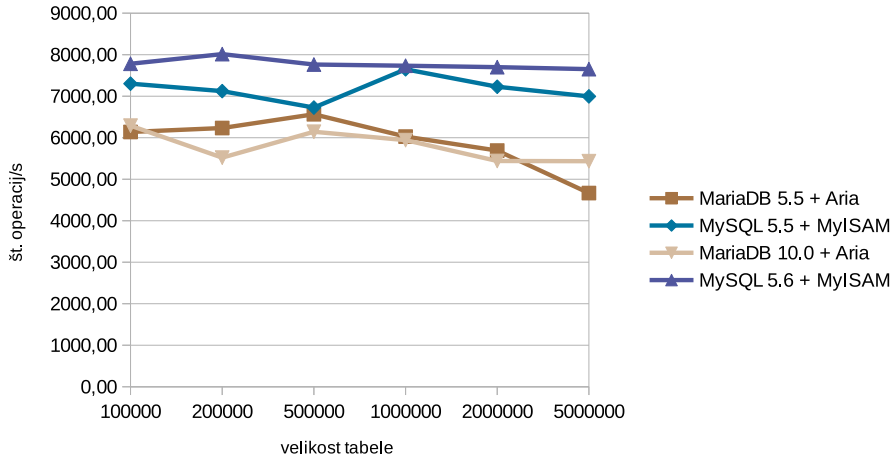
Posodabljanje stolpca v vrstici (UPDATE)

Kot zadnjo od osnovnih operacij smo testirali posodabljanje vrednosti stolpca v tabeli. Tu ločimo dva primera, in sicer posodobitev stolpca, ki je ključ (indeks) in posodobitev "navadnega", neideksiranega stolpca.

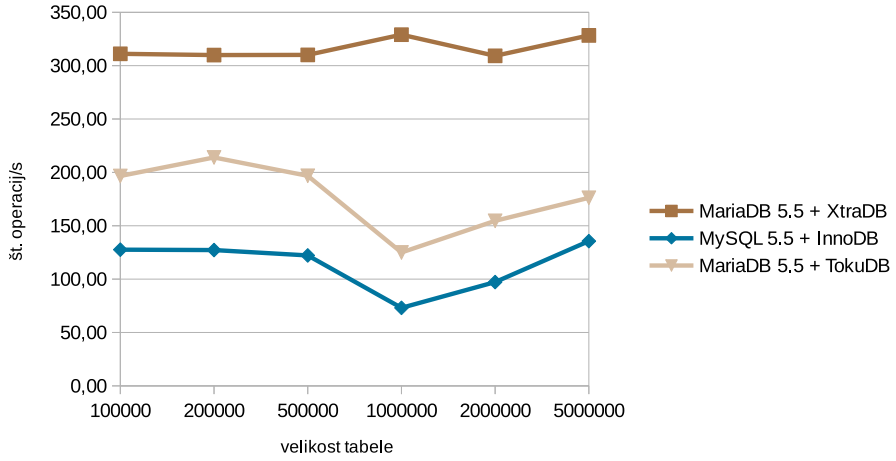
Neindeksiran stolpec

Na sliki 5.12 so prikazani rezultati testiranja performans za posodobitev stolpca na netransakcijskih shranjevalnih mehanizmih. Na prvem mestu je z 7653–8013 posodobitvami v sekundi MyISAM v novejšem MySQL, za njim pa njegova starejša verzija. Aria dosega v povprečju 77 % vrednosti MyISAM, kot že v nekaterih primerih prej pa so te nižje v novejšem sistemu MariaDB.

Od transakcijskih shranjevalnih mehanizmov, za katere so rezultati na slikah 5.13 in 5.14, je ponovno najboljša Aria – a le pri manj nitih. Kot pri brisanju in vstavljanju tudi tu v primeru z 8 in 16 nitmi zaostane za ostalimi shranjevalnimi mehanizmi. Od teh sicer v starejših sistemih močno vodi XtraDB, ki vzdržuje med 309 in 329 operacij na sekundo, okrog 60 % njegove zmogljivosti dosega TokuDB, več kot pol slabši pa je InnoDB na zadnjem mestu.

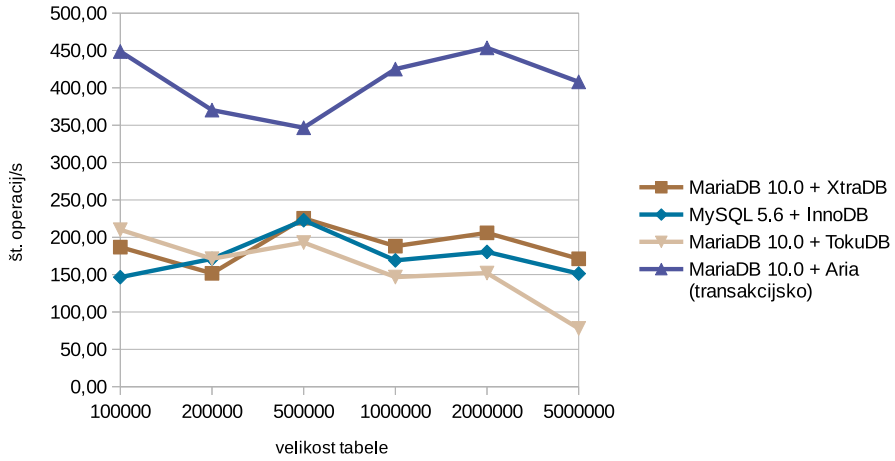


Slika 5.12: Število posodobitev stolpca v eni sekundi v netransakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB



Slika 5.13: Število posodobitev stolpca v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB 5.5.40

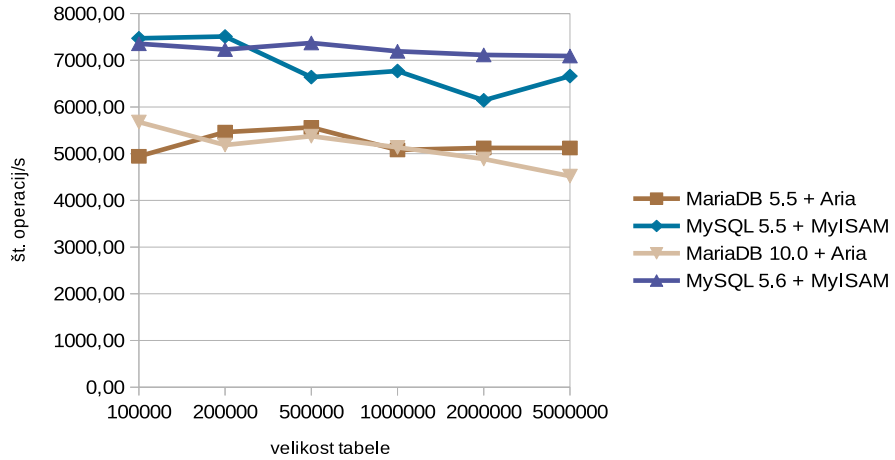
V novejših sistemih je, izvzemši Ario, pri manjših tabelah najhitrejši TokuDB, od 500.000-vrstične tabele naprej pa XtraDB. Krivulji slednjega je sicer zelo podobna krivulja InnoDB, le postavljena je nekoliko niže.



Slika 5.14: Število posodobitev stolpca v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL 5.6 in MariaDB 10.0

Indeksiran stolpec

V drugem testu je bil posodobljen indeksiran stolpec `k`. Zaradi zahtevnejše obravnave takšnih stolpcev smo pričakovali nekoliko višji čas izvedbe oziroma manj izvedenih operacij v eni sekundi.

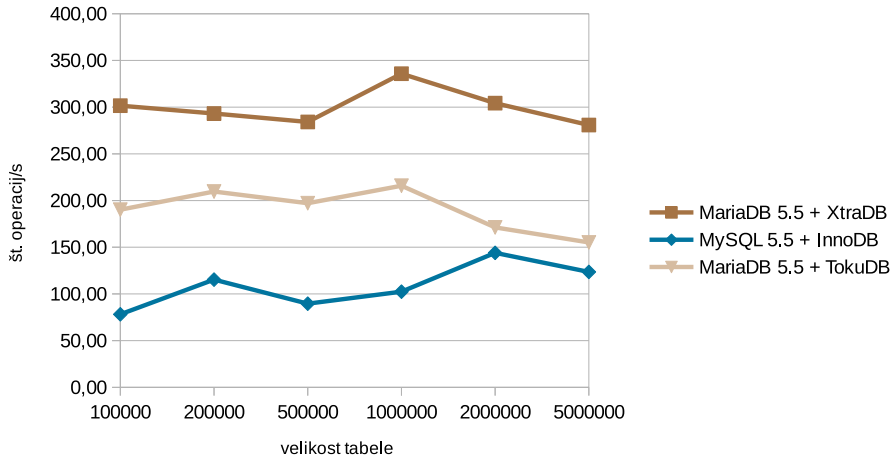


Slika 5.15: Število posodobitev indeksa v eni sekundi v netransakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB

Graf na sliki 5.15 prikazuje izmerjene vrednosti na netransakcijskih shranjevalnih mehanizmih MyISAM in Aria. Pri dveh najmanjših tabelah je za 1,5 % oziroma

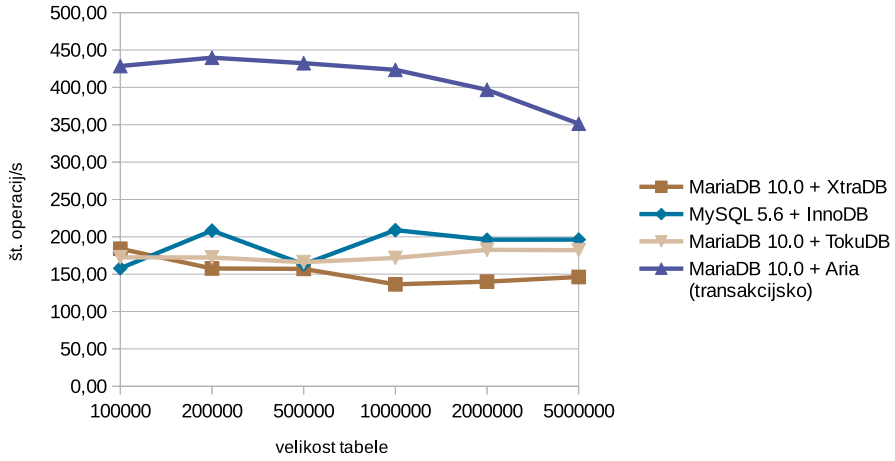
3,7 % boljši starejši MyISAM, sicer pa prevladuje novejši, katerega krivulja je tudi bolj enakomerna, giblje se med 7091 in 7355 operacij/s. Performanse Arie so boljše v starejšem sistemu MariaDB, dosežene vrednosti so med 4944 in 5564 operacij/s, kar je v povprečju 72 % najvišjih (MyISAM-ovih).

Na slikah 5.16 in 5.17 so rezultati testa na transakcijskih shranjevalnih mehanizmih, na prvi v sistemih 5.5, na drugi pa v novejših različicah. Kot pri vseh pisalnih operacijah doslej tudi tu najvišje vrednosti dosega XtraDB v MariaDB 5.5: med 281 in 336 operacij/s, medtem ko novejša različica izvede le 41–61 % toliko. InnoDB, ki je v starejšem sistemu na zadnjem mestu, tudi tu kaže izboljšanje, saj je, razen pri najmanjši tabeli, od polno transakcijskih shranjevalnih mehanizmov v novejših SUPB najboljši s 196–209 operacijami/s. Še vedno pa s 351–441 posodobitvami v sekundi vodi Aria, ki pa (tako kot v ostalih primerih) zaostane pri več kot 4 nitih.



Slika 5.16: Število posodobitev indeksa v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL in MariaDB 5.5.40

Na osnovi rezultatov v tem poglavju lahko rečemo, da je MyISAM v novem MySQL nedvomno izboljšan – v vseh 5 testih je med netransakcijskimi shranjevalnimi mehanizmi najhitrejši. Najverjetneje lahko v prihajajoči verziji 5.7 pričakujemo še večjo zmogljivost in krajše odzivne čase. Nasprotno je Aria v novem MariaDB nekoliko počasnejša; razvijalci sicer ne skrivajo, da je razvoj naslednje različice,



Slika 5.17: Število posodobitev indeksa v eni sekundi v transakcijskih shranjevalnih mehanizmih v sistemih MySQL 5.6 in MariaDB 10.0

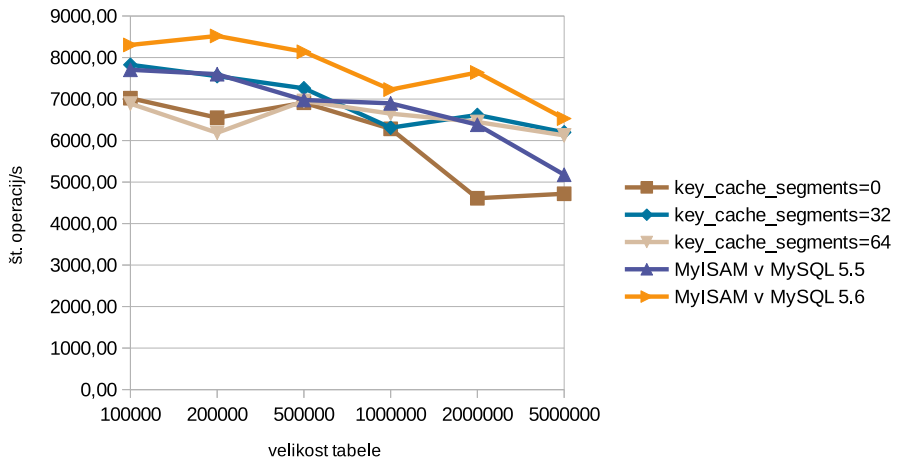
2.0, potisnjen na stranski tir, ker se posvečajo izboljšanju celotnega sistema [30].

Med polnotransakcijskimi shranjevalnimi mehanizmi je, razen pri poizvedovanju, kjer vodi InnoDB v MySQL 5.6, na prvem mestu XtraDB v, nekoliko presenetljivo, starejši različici MariaDB. Glede na to, da sta novi verziji InnoDB in XtraDB, 5.6.22, vključeni v naslednjo, trenutno najnovejšo različico MariaDB 10.0.16, bi bili rezultati primerjave s to različico verjetno bolj relevantni. Sodeč po številu operacij, ki jih v sekundi izvede transakcijska Aria, pa je ta precej obetavna, dokler imamo malo odjemalcev in visoke sočasnosti ne potrebujemo.

5.2 Testiranje segmentiranega predpomnilnika ključev

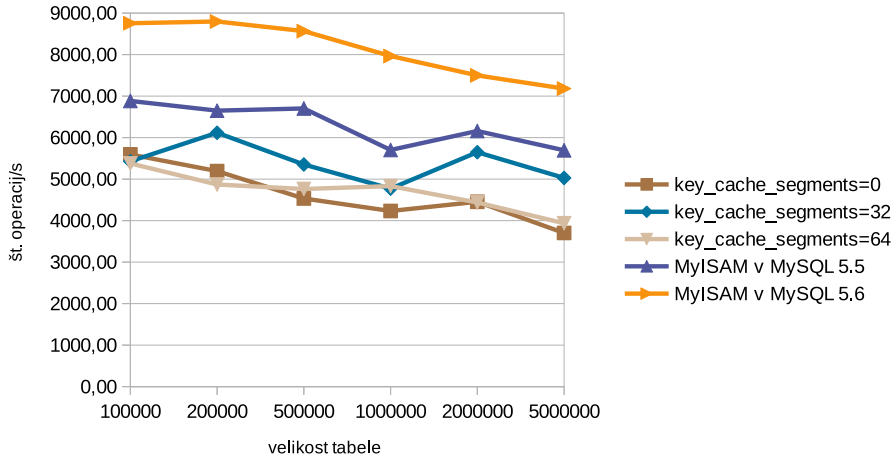
S testoma `select_random_points` in `select_random_ranges` smo želeli preveriti vpliv segmentacije predpomnilnika ključev za shranjevalni mehanizem MyISAM na zmogljivost oziroma prepustnost pri sočasnih dostopih. Testa smo izvedli z običajnim, nesegmentiranim predpomnilnikom, tj. z vrednostjo spremenljivke `key_cache_segments=0`, ter z 32 in 64 segmenti. Prikazani so rezultati obeh testov v sistemu MariaDB 10.0; za primerjavo so na grafih tudi rezultati testov na MyISAM v sistemih MySQL.

Najprej poglejmo izmerjene vrednosti za prvo poizvedbo, pri kateri se generira množica 10 števil, na sliki 5.18. Če za trenutek izvzamemo MyISAM v MySQL, so razen pri tabeli z milijonom vrstic poizvedbe najhitrejše z na 32 segmentov razdeljenim predpomnilnikom ključev. Nesegmentiran in najbolj segmentiran sta pri manjših tabelah po zmogljivosti precej blizu skupaj, pri večjih pa prepustnost z nesegmentiranim predpomnilnikom ključev precej upade; število poizvedb je s segmentiranim predpomnilnikom večje za 31 oz. 43 %. V večini primerov se izvede v eni sekundi več operacij kot z MyISAM v starejši verziji MySQL, še vedno pa s 5–10 % višjimi vrednostmi vodi MyISAM v MySQL 5.6.



Slika 5.18: Doseženo število poizvedb 10 naključnih vrstic v eni sekundi s shranjevalnim mehanizmom MyISAM v sistemu MariaDB 10.0 pri 16 nitih in različnem številu segmentov predpomnilnika ključev

Pri drugem testu, kjer poizvedujemo v okviru 10 razponov, je situacija (slika 5.19) podobna. Znova sta najvišje MyISAM v novejšem in MyISAM v starejšem MySQL, sledi pa MyISAM v MariaDB z 32-segmentnim predpomnilnikom ključev. Pohitritev je tu v primerjavi z nesegmentiranim predpomnilnikom 13- do 36-odstotna. Na testih performans segmentiranega predpomnilnika ključev pri MariaDB so te sicer s 64-segmentnim predpomnilnikom izboljšane za celo 270 % vrednosti, doseženih z nesegmentiranim predpomnilnikom ključev [62].



Slika 5.19: Doseženo število poizvedb 10 naključnih razponov v eni sekundi s shranjevalnim mehanizmom MyISAM v sistemu MariaDB 10.0 pri 16 nitih in različnem številu segmentov predpomnilnika ključev

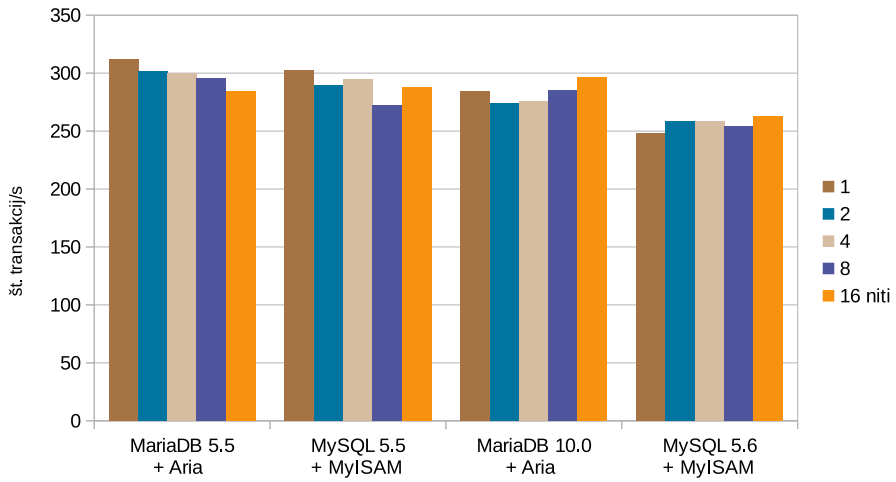
Naši rezultati sicer niso pokazali tolikšne pohitritve, vseeno pa lahko rečemo, da je pri tako velikih tabelah in takšnih poizvedbah za naš sistem 32 segmentov v predpomnilniku ključev najboljša izbira – zelo verjetno pa bi z nadaljnjim preizkušanjem lahko našli še ustreznejšo, pri obširnejših poizvedbah in z več sočasnimi odjemalci pa bi se morda lahko približali "uradnim" rezultatom iz MariaDB.

5.3 Transakcije (OLTP)

Pri tem testu nas je zanimala transakcijska zmogljivost, tj. kolikokrat se zaporedje 20 SQL-stavkov izvrši v eni sekundi. Na slikah 5.20 in 5.21 prikazujemo rezultate za 500.000 vrstic veliko tabelo. Kot vidimo, pri netransakcijskih vodi

Aria v starejši različici MariaDB, razen pri 16 nitih, kjer nekaj "transakcij" več dosežeta Aria v MariaDB 10.0 in MyISAM v MySQL 5.6. Zanimivo je, da oba shranjevalna mehanizma dosegata v sistemih 5.5 višje vrednosti kot v novejših. Ariine se gibljejo med 284 in 312 izvedenih transakcij/s, MyISAM-ove pa med 272 in 302 transakcijama/s.

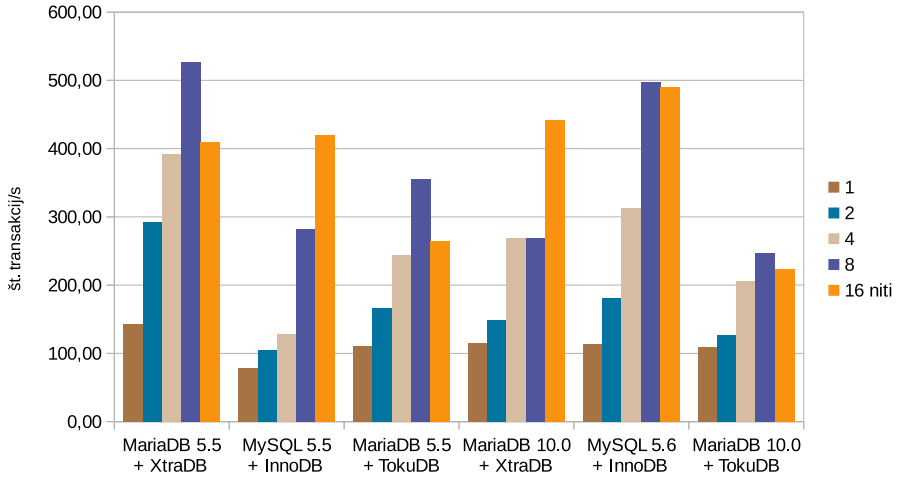
Če te vrednosti primerjamo s tistimi iz transakcijskih shranjevalnih sistemov, opazimo, da so slednje vsaj pri izvedbi z 1 nitjo precej manjše. Vendar pa se v primerih z 8 ali s 16 nitmi povzpnejo tudi do 1,7-kratnika vrednosti netransakcijskih shranjevalnih mehanizmov, največ transakcij v eni sekundi, 527, je izvedel XtraDB v MariaDB 5.5 pri 8 nitih, pri 16 pa je bil s 490 hitrejši InnoDB v MySQL 5.6. Tudi pri ostalih številih niti je sicer hitrejši XtraDB v MariaDB 5.5, v novejši verziji pa so vrednosti nekoliko manjše. Tudi rezultati TokuDB so v 5.5 boljši kot v 10.0.



Slika 5.20: Doseženo število "transakcij" v eni sekundi v shranjevalnih mehanizmih MyISAM in Aria

5.4 Kontrolna vsota (CHECKSUM)

Na slikah 5.22 in 5.23 so predstavljeni rezultati meritev časa, potrebnega za izračun kontrolne vsote tabele (operacija `CHECKSUM TABLE`) z 1 nitjo; na prvi sliki iz sistemov različice 5.5, na drugi pa iz 10.0 oziroma 5.6.



Slika 5.21: Doseženo število transakcij v eni sekundi v transakcijskih shranjevalnih mehanizmih

Čas izvedbe v vseh shranjevalnih mehanizmih in sistemih narašča skoraj linearno z velikostjo tabele, nekoliko odstopa le TokuDB pri 20.000 vrstic veliki tabeli, kjer smerni koeficient poskoči z okrog 1,2 na 1,5.

V obeh primerih so rezultati shranjevalnih mehanizmov InnoDB in XtraDB blizu skupaj, InnoDB v MySQL je povečini nekoliko – v povprečju za 5 % – hitrejši. V novejših sistemih je ta razlika še večja, 15-odstotna, na račun počasnejšega XtraDB.

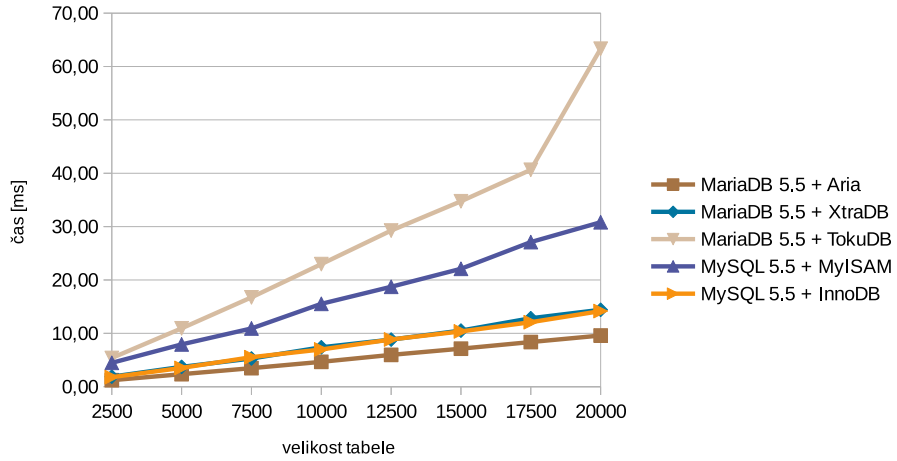
Shranjevalni mehanizem Aria medtem za takšno operacijo potrebuje le okrog 68 %, v MariaDB 10.0 pa celo samo 64 % časa, ki ga porabi InnoDB. MyISAM-ovi časi so približno 3-krat daljši od Ariinih, TokuDB-jevi pa celo do 6,5-krat.

Na podlagi teh rezultatov lahko zaključimo, da je obljuba o hitrejši izvedbi kontrolne vsote z Ario vsekakor izpolnjena.

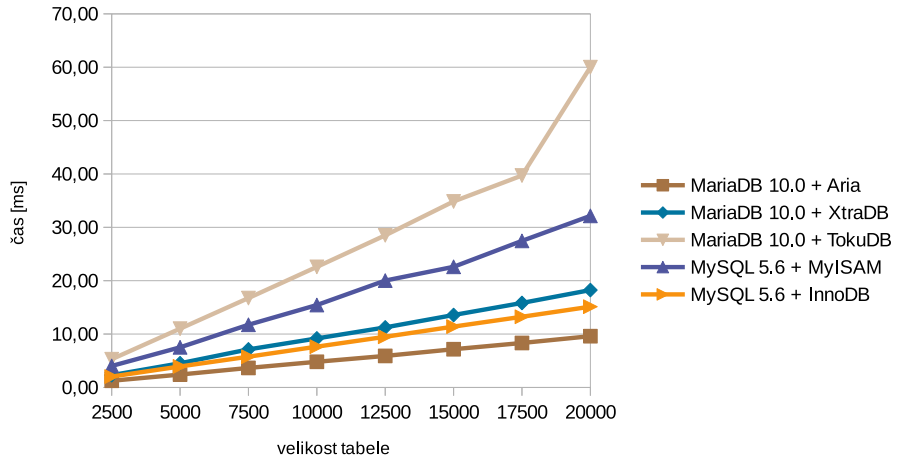
5.5 Pretvorba nabora znakov

Rezultati tega testa, za 200.000 vrstic veliko tabelo prikazani na slikah 5.24 (starejša SUPB) in 5.25 (novejša SUPB), kažejo izboljšanje vseh shranjevalnih mehanizmov v novejših različicah SUPB.

Razlika je največja v InnoDB, ki je v novem MySQL hitrejši med 6,75-krat (pri 16 nitih) in 15,8-krat (pri eni niti), najmanjša pa v MyISAM (3,8–6,9), ki je najhi-



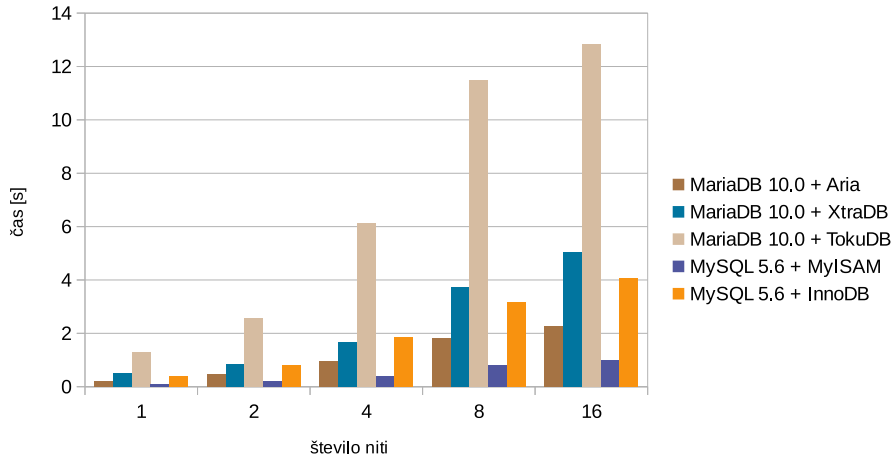
Slika 5.22: Povprečni časi za izračun kontrolne vsote tabele v sistemih MariaDB in MySQL 5.5



Slika 5.23: Povprečni časi za izračun kontrolne vsote tabele v sistemih MariaDB 10.0 in MySQL 5.6

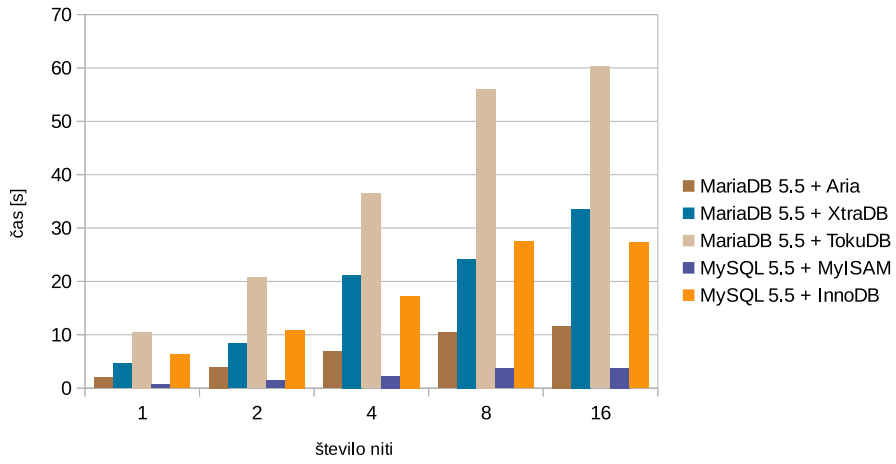
trejši tako v starejših kot v novejših SUPB. Pri ostalih shranjevalnih mehanizmih se faktor pohitritve sicer giblje med 4,7 in 9,4.

Kot že rečeno, je najhitrejši MyISAM (v MySQL 5.6), za njim pa z 2-kratniki njegovih vrednosti Aria (v MariaDB 10.0). V primerjavi z različico 5.5.40 traja pretvorba nabora znakov tabele v Arii v novi različici samo 11–19 % prejšnjih vrednosti. Tako lahko tudi za to operacijo potrdimo pohitritev, ki jo navajajo



Slika 5.24: Povprečni časi za pretvorbo nabora znakov tabele z 200.000 vrsticami v sistemih MariaDB in MySQL 5.5

pri MariaDB, prav tako pa je operacija v novejših sistemih hitrejša tudi pri vseh ostalih shranjevalnih mehanizmih. Daleč najpočasnejši je sicer TokuDB, katerega časi izvedbe operacije so v MariaDB 10.0 6-krat daljši kot Ariini.



Slika 5.25: Povprečni časi za pretvorbo nabora znakov tabele z 200.000 vrsticami v sistemih MariaDB 10.0 in MySQL 5.6

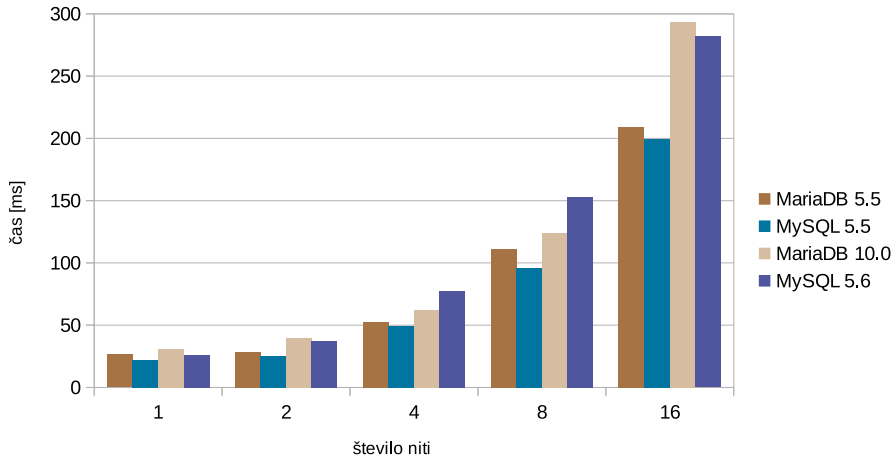
5.6 Testiranje optimizatorja poizvedb

Teste `innerjoin`, `groupby` in `subquery` smo pripravili in izvedli za preizkus optimizatorja poizvedb v MariaDB. Izvedeni so bili na vzorčni podatkovni bazi Sakila, katere tabele uporabljajo shranjevalni mehanizem InnoDB, ki ga v sistemu MariaDB zamenjuje XtraDB.

JOIN

Na sliki 5.26 so prikazani povprečni časi, v katerih se je izvedla poizvedba z združitvijo (`INNER JOIN`, gl. razdelek 4.5). V vseh primerih je najhitrejši InnoDB v MySQL 5.5, največ časa za poizvedbo pa porabi MariaDB 10.0 (pri 1 in 2 nitih) oziroma MySQL 5.6 (pri ostalih številih niti). Razmerje med najpočasnejšim in najhitrejšim je med 1,38 in 1,59; v povprečju prvi torej za poizvedbo porabi še polovico časa, v katerem slednji, tj. MySQL 5.5, vrne rezultat. Ta vrednost je pri 1 niti 22,34 ms, pri 4 nitih 49,28 s, pri 16 pa 199,84 s.

Razlike v višini stolpcev na diagramu niso zelo velike, kljub temu pa rezultat preseneča, saj sta pri obeh sistemih novejši različici počasnejši. Da je optimizator poizvedb v MariaDB izboljšan in poizvedbe zato hitrejše, na osnovi teh rezultatov ne moremo zaključiti.



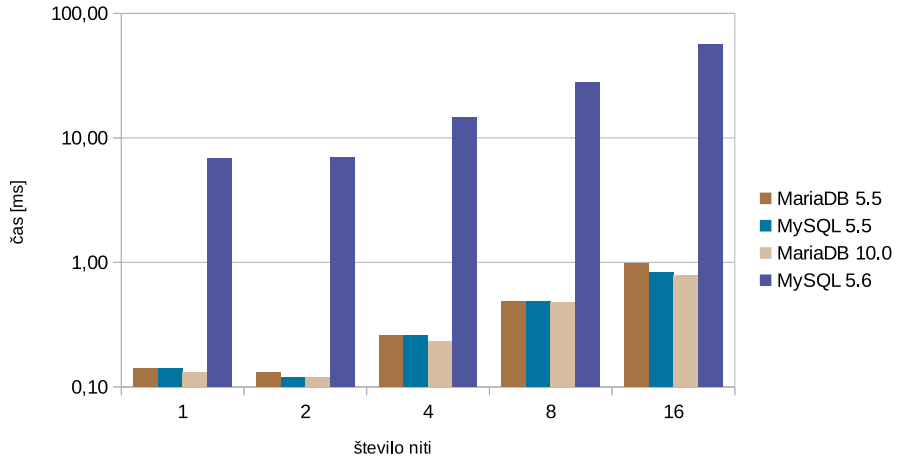
Slika 5.26: Povprečni časi za poizvedbo s tremi združitvami INNER JOIN v sistemih MariaDB z XtraDB in v MySQL z InnoDB

GROUP BY

Rezultati drugega testa, ki vsebuje poizvedbo z agregacijsko funkcijo GROUP BY, so prikazani na sliki 5.27. Kot vidimo, so časi poizvedbe v sistemih 5.5 in v MariaDB 10.0 večinoma zelo blizu eden drugemu, medtem ko v MySQL novejše različice skokovito narastejo. Rezultat je presenetljiv, saj naj bi bila tako shranjevalni mehanizem kot sam sistem MySQL z novejšo izdajo precej hitrejša in izboljšana. Poizvedba v MySQL 5.6 potrebuje med 49- in 58-krat toliko časa kot drugi najpočasnejši, XtraDB v MariaDB 5.5 oz. InnoDB v MySQL 5.5.

Najboljši rezultat sicer pri vseh nitih dosega XtraDB v MariaDB 10.0; z enim odjemalcem poizvedbo opravi v 0,11 ms, s 16 pa v 0,74 ms. MariaDB 5.5 in MySQL 5.5 sta bodisi izenačena ali pa je slednji malce hitrejši.

Za razliko od prejšnjega testa ta govori v prid MariaDB in obljubljeni izboljšavi optimizatorja poizvedb. V določeni meri so rezultati podobni rezultatom testiranja zmogljivosti optimizatorja v MariaDB 5.3, objavljenimi na blogu MariaDB, kjer MySQL 5.6 porabi tudi do 5,6-krat toliko časa kot MariaDB, MySQL 5.5 pa celo do 31,5-krat toliko (test je bil sicer izveden s kompleksnimi poizvedbami z združevanjem) [61].

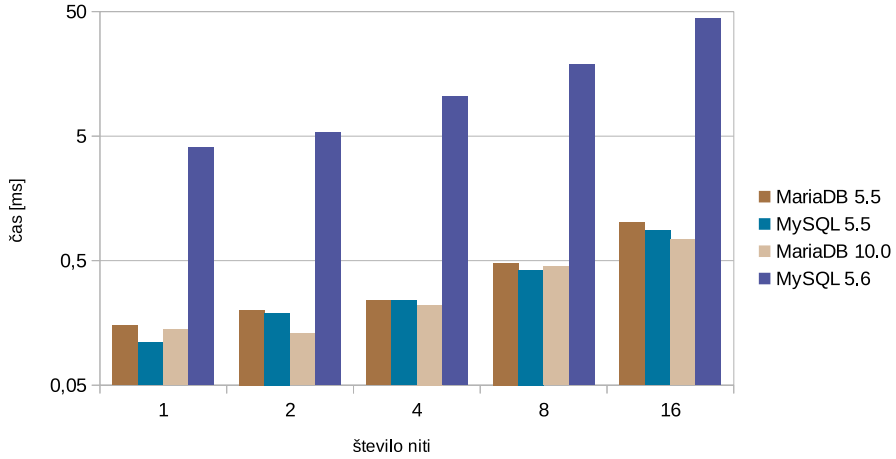


Slika 5.27: Povprečni časi za poizvedbo z agregacijsko funkcijo GROUP BY v sistemih MariaDB z XtraDB in v MySQL z InnoDB

Gnezdena poizvedba

Rezultati zadnjega testa v tem sklopu so prikazani z diagramom na sliki 5.28. Tudi tu izstopa MySQL 5.6, katerega izvajalni časi so precej daljši kot v MySQL 5.5 in MariaDB. Količnik z drugim najpočasnejšim, XtraDB v MariaDB 5.5, je tu sicer manjši kot v prejšnjem testu, med 27 in 44.

Najmanj časa za posamezno poizvedbo (0,11 ms) je sicer pri izvedbi z 1 nitjo potreboval MySQL 5.5, medtem ko v večini ostalih primerov zmaguje MariaDB 10.0 (0,22 ms pri 4 nitih, 0,74 ms pri 16 nitih). Tako kot pri prejšnjem testu lahko tudi tu potrdimo izboljšan optimizator in hitrejše izvajanje kompleksnih poizvedb.



Slika 5.28: Povprečni časi za izvedbo trojne gnezdene poizvedbe v sistemih MariaDB z XtraDB in v MySQL z InnoDB

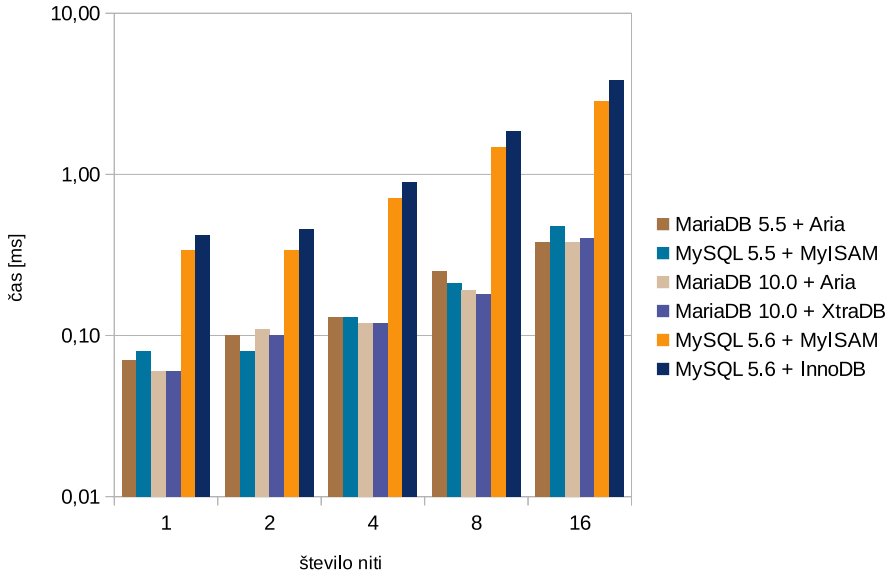
S temi tremi testi smo želeli izmeriti performanse zahtevnejših poizvedb in preveriti obljubljene pohitritve. V MariaDB navajajo mnogo izboljšav v optimizatorju, med drugim predpomnilnik gnezdenih poizvedb [63], katerega prisotnost bi lahko pojasnila velike razlike v časih, ki jih za takšne poizvedbe potrebuje MySQL. Kljub temu pa ostaja vprašanje, kaj je razlog za tolikšno upočasnitev v novejši različici v primerjavi s starejšo.

Z dodatnimi testi in še zapletenejšemi poizvedbami bi izpopolnjen optimizator lahko nadalje preverili, nedvomno pa lahko potrdimo, da na tem področju sistemu MariaDB dobro kaže – sploh v primerjavi z novim MySQL.

5.7 Polnotekstovno iskanje

Poizvedba s polnotekstovnim iskanjem je bila izvedena v tabeli s shranjevalnim mehanizmom MyISAM (MySQL) oziroma Aria (MariaDB), v novejših izdajah pa tudi z InnoDB (MySQL) oziroma XtraDB (MariaDB).

Najhitreje so bile večinoma izvedene poizvedbe v tabelah z XtraDB in Ario (v MariaDB 10.0); ustrezna stolpca na sliki 5.29 sta poravnana ali pa je razlika med njima zelo majhna. Pri dveh nitih je sicer nekoliko hitrejši MySQL 5.5 s shranjevalnim mehanizmom MyISAM – povprečni čas izvedbe je bil 0,8 ms.



Slika 5.29: Povprečni časi izvedbe polnotekstovnega iskanja

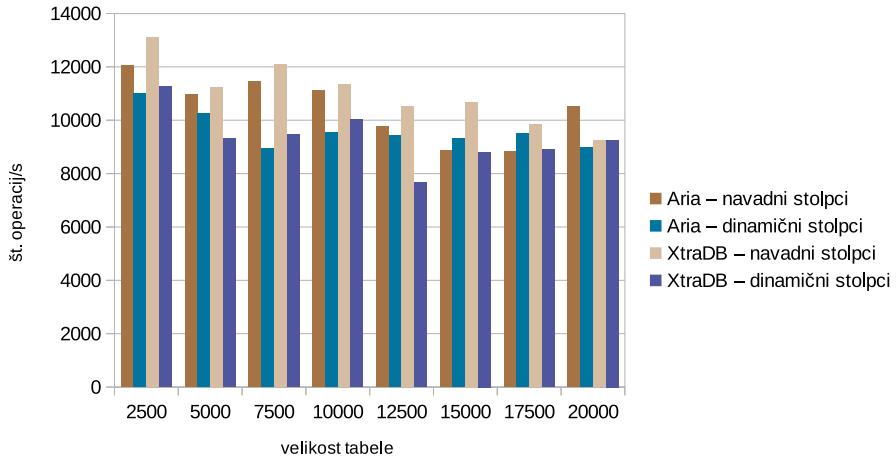
Če vzorčno podatkovno bazo Sakila hranimo na sistemu MariaDB 10.0, bodo polnotekstovna iskanja po naslovih in opisih filmov potrebovala 6- do 10-krat manj časa kot v sistemu MySQL 5.6, hitrejša pa so tudi v primerjavi z MyISAM in Ario v starejšem MariaDB. InnoDB oz. XtraDB torej ne samo da podpirata polnotekstovne indekse, temveč je iskanje z njimi, po naših meritvah sodeč, lahko (v primeru MariaDB) celo hitrejše kot v MyISAM ali Arii.

5.8 Dinamični in navidezni stolpci

Zadnja testa sta nastala po pregledu ”nestandardnih” dinamičnih in navideznih stolpcev, saj nas je zanimala tudi performačna plat njihove uporabe. Ker so dinamični stolpci dodatna struktura v obstoječem ”navadnem” stolpcu, smo predvidevali, da bodo poizvedbe, ki jih vključujejo, potrebovale več časa kot poizvedbe po običajnih stolpcih. V drugem testu, kjer ena poizvedba vključuje štiri navidezne stolpce, druga pa običajne in izračun med poizvedbo, smo ravno tako pričakovali daljši čas za prvo, saj vključuje en izračun več (dva navidezna stolpca v primerjavi z enim izračunom).

Testa smo izvedli v MariaDB 10.0 z XtraDB in Ario; rezultati meritev prvega, z dinamičnimi stolpci, so na sliki 5.30, drugega pa na sliki 5.31.

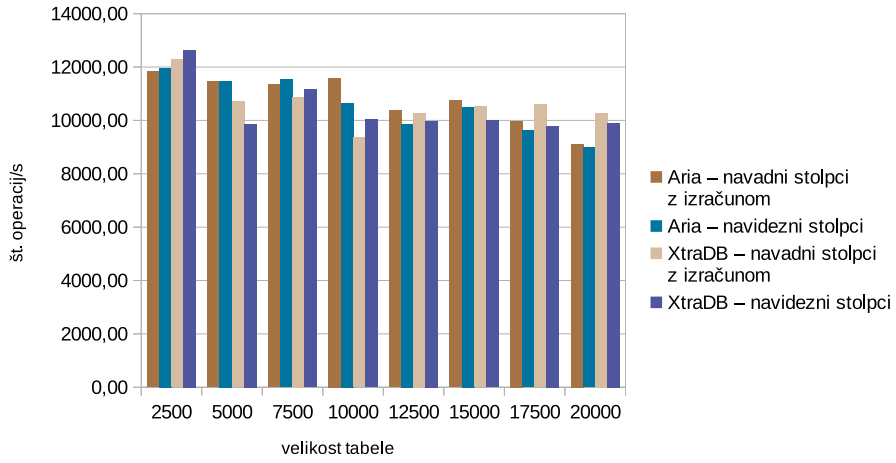
V skladu s pričakovanji smo z uporabo dinamičnih stolpcev dosegli manj operacij v eni sekundi kot z navadnimi stolpci; v Arii razen pri dveh velikostih tabel, kjer je bilo z dinamičnimi stolpci izvedenih za 5 oz. 7 % več poizvedb, v povprečju 7 % manj, v XtraDB pa celo za 15 % manj poizvedb kot z uporabo navadnih stolpcev. Dinamični stolpci so v Arii torej hitrejši kot v XtraDB.



Slika 5.30: Doseženo število poizvedb z navadnimi in dinamičnimi stolpci

V drugem testu, kjer smo v enem scenariju v poizvedbi pridobili vrednosti navadnih stolpcev oziroma stolpca, pomnoženega z vrednostjo, v drugem pa vrednosti navideznih stolpcev, so razlike manjše. Pri obeh shranjevalnih mehanizmih se je pri treh velikostih tabele izvedlo nekaj več poizvedb v primeru z navideznimi

stolpci, sicer pa jih je bilo v povprečju za 2 % manj.



Slika 5.31: Doseženo število poizvedb z navadnimi stolpci in izračunom ob poizvedbi ter z navideznimi stolpci

Uporaba dinamičnih stolpcev se glede na rezultate performančno bolj izplača v shranjevalnem mehanizmu Aria. Razlika med številom poizvedb z navadnimi ali dinamičnimi stolpci tu ni zelo velika, prav tako ne v primeru uporabe navideznih stolpcev – v obeh shranjevalnih mehanizmih. Menimo, da bi bila kljub nekoliko slabšim performansam za naš primer, kjer atributi zapisov v tabeli niso vnaprej določeni, uporaba dinamičnih (in navideznih) stolpcev bolj primerna in bi strukturno odtehtala nekoliko daljše čase poizvedbe.

Poglavje 6

Sklepne ugotovitve

MariaDB obljublja veliko; seznami vtičnikov, zmožnosti in izboljšav so dolgi in ob pregledovanju dokumentacije ter številnih člankov in vodičev je bilo težko narediti izbor tistih, ki jih bomo vključili v diplomsko nalogo. Z opisi vseh funkcionalnosti in primeri njihove uporabe bi se lahko razpisali še na mnogih straneh. Ena bolj zanimivih so nedvomno dinamični stolpci, še posebej z omogočenjem niza (namesto števil) za ime stolpca in s funkcijo, ki imena stolpcev in njihove vrednosti vrne v obliki JSON – tudi za gnezdene stolpce –, kar bo privlačno za mnoge razvijalce, saj so podatki že v strukturirani obliki in pripravljeni za nadaljnjo obdelavo.

Dobrodošli pridobitvi sta NoSQL-vtičnika v obeh sistemih; oba ponujata dovolj enostaven vmesnik in kratke odzivne čase, na razpolago pa so tudi mnoge knjižnice in API-ji, s katerimi je NoSQL-komunikacija z SQL-sistemom preko obeh še lažja.

Pri namestitvi in uporabi enega in drugega sistema sicer nismo naleteli na večje razlike v samem vmesniku in obnašanju. Uporabniku ukazne vrstice prijazen dodatek v MariaDB je med oglatimi oklepaji prikazano ime podatkovne baze, ki je trenutno izbrana in v kateri izvajamo operacije. Nekoliko enostavnejša je bila tudi namestitev novejše različice oziroma nadgradnja (s 5.5 na 10.0), saj repozitorij MySQL ni vseboval najnovejše različice 5.6.22, ki smo jo zato morali namestiti ročno s paketi .deb. Ob poskusu namestitve razvojne (*labs*) različice pa smo naleteli na nezdružljivost z nekaterimi knjižnicami, zaradi česar ni delovalo orodje sysbench in smo namestitev opustili.

Rezultati zmogljivostnih testov so nekoliko presenetili; z izjemo InnoDB je prepustnost v transakcijskih shranjevalnih mehanizmih v novejših sistemih v pri-

merjavi s starejšimi upadla. V MySQL 5.6 sta sicer izboljšana tako MyISAM kot InnoDB, s čimer Oracle kljubuje obtožbam, da razvoj stoji, in postavlja pod vprašaj zmogljivost izpeljanke MariaDB in alternativnih shranjevalnih mehanizmov, kot sta XtraDB in TokuDB. Performanse slednjih sicer v naših testih niso dosegle pričakovanj, v mnogih primerih so se izkazale za celo slabše od performans "osnovnega" InnoDB, vendar pa moramo upoštevati, da sta oba shranjevalna mehanizma v osnovi namenjena in optimizirana za bolj obremenjene sisteme; za boljšo sliko in relevantnejše rezultate bi ju morali testirati z več hkratnimi odjemalci in/ali večjimi tabelami. Prav tako imajo vsi shranjevalni mehanizmi (in oba sistema) kopico lastnih parametrov in spremenljivk; z nastavitvijo na vrednosti, ki odražajo potrebe dejanskega okolja, bi se najverjetneje približali boljšim, želenim rezultatom.

Posebej se moramo dotakniti Arie, ki je nekako osrednja komponenta, okrog katere je bil sistem MariaDB postavljen in ki jo v prihodnosti čaka še mnogo dodelav in izboljšav. Performančno še ne dosega shranjevalnega mehanizma MyISAM, je pa nedvomno obetavna zaradi sposobnosti obnovitve oziroma povrnitve ob morebitnem sesutju, še bolj pa zaradi občutno večjega števila v sekundi izvedenih operacij v primerjavi s polno transakcijskimi shranjevalnimi mehanizmi, kadar imamo sočasnih zahtev malo. Če oziroma ko bodo njene načrtovane zmožnosti, tj. skladnost z ACID in MVCC ter sočasnost pisalnih operacij, realizirane, zna biti močna konkurenca obstoječim shranjevalnim mehanizmom.

Sistemu MariaDB v prid govorijo mnoge nove zmožnosti in dodatni shranjevalni mehanizmi, aktivna in odzivna skupina razvijalcev, ki na spletni strani redno objavlja prispevke in rezultate testov zmogljivosti, obširna, posodobljena in pregledna dokumentacija, enostavnost namestitve (in nadgradnje), nenazadnje pa tudi obljuba o delovanju s skupnostjo – in zanjo – ter o ohranjanju odprtosti kode celotnega paketa. Slednje zna pri marsikom pretehtati, še posebej, ker splošno dostopne izdaje MySQL 5.7, ki sicer obeta mnogo izboljšav in novosti, zaenkrat še ni na vidiku.

Literatura

[1] The State of the Open Source Database Market: MySQL Leads the Way. Dostopano 7. 10. 2014 na: http://www.scalebase.com/the-state-of-the-open-source-database-market-mysql-leads-the-way/

[2] DB-Engines. Dostopano 12. 2. 2015 na: http://db-engines.com/en/

[3] (2011) Forta, B. MariaDB Crash Course. Addison-Wesley Professional, 2011. ISBN 978-0-321-79994-4.

[4] Release Notes, MariaDB 5.1 Series. Dostopano 7. 10. 2014 na: https://mariadb.com/kb/en/mariadb/development/release-notes/release-notes-mariadb-51-series/

[5] MariaDB. Dostopano 3. 2. 2015 na: http://en.wikipedia.org/wiki/MariaDB

[6] MariaDB Case Studies. Dostopano 18. 2. 2015 na: https://mariadb.com/kb/en/mariadb/case-studies/

[7] Distributions Which Include MariaDB. Dostopano 3. 2. 2015 na: https://mariadb.com/kb/en/mariadb/distributions-which-include-mariadb/

[8] A look at the MySQL forks. Dostopano 4. 2. 2015 na: http://lwn.net/Articles/329626/

[9] MySQL and the forks in the road. Dostopano 4. 2. 2015 na: http://www.h-online.com/open/features/MySQL-and-the-forks-in-the-road-1829242.html

[10] Percona Server. Dostopano 29. 10. 2014 na:
http://www.percona.com/software/percona-server

[11] Percona Server 5.6. Dostopano 29. 10. 2014 na:
http://www.percona.com/software/percona-server/ps-5.6

[12] Users of Percona Server. Dostopano 2. 11. 2014 na:
http://www.percona.com/software/percona-server/users

[13] Drizzle7: The First GA release of the Drizzle Project. Dostopano 15. 11. 2014 na: http://www.rackspace.com/blog/drizzle7-the-first-ga-release-of-the-drizzle-project/

[14] Drizzle Download. Dostopano 14. 11. 2014 na:
http://www.drizzle.org/content/download

[15] Monty says: Time to move on. Dostopano 21. 2. 2015 na:
http://monty-says.blogspot.com/2009/02/time-to-move-on.html

[16] Monty says: Help saving MySQL. Dostopano 21. 2. 2015 na:
http://monty-says.blogspot.com/2009/12/help-saving-mysql.html

[17] About the MariaDB Foundation. Dostopano 21. 2. 2015 na:
https://mariadb.org/en/foundation/

[18] Why SkySQL becoming MariaDB Corporation will be good for the MariaDB Foundation. Dostopano 21. 2. 2015 na:
https://blog.mariadb.org/why-skysql-becoming-mariadb-corporation-will-be-good-for-the-mariadb-foundation/

[19] (2012) Bartholomew, D.: MariaDB vs. MySQL. Dostopano 7. 10. 2014 na:
http://mariadb.com/sites/default/files/MariaDB

[20] MariaDB roadmap. Dostopano 28. 10. 2014 na:
https://mariadb.com/kb/en/mariadb/development/general-development-information/development-plans/mariadb-roadmap/

[21] Explanation on MariaDB 10.0. Dostopano 23. 1. 2015 na:
http://blog.mariadb.org/explanation-on-mariadb-10-0/

[22] MariaDB 10.1.0 Release Notes. Dostopano 16. 2. 2015 na: https://mariadb.com/kb/en/mariadb/mariadb-1010-release-notes/

[23] System variable differences between MariaDB 5.5 and MySQL 5.5. Dostopano 23. 1. 2015 na: https://mariadb.com/kb/en/mariadb/system-variable-differences-between-mariadb-55-and-mysql-55/

[24] System variable differences between MariaDB 10.0 and MySQL 5.6. Dostopano 23. 1. 2015 na: https://mariadb.com/kb/en/mariadb/system-variable-differences-between-mariadb-100-and-mysql-56/

[25] MariaDB versus MySQL - features. Dostopano 27. 1. 2015 na: https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/

[26] What's New in the MySQL 5.6 Release Candidate. Dostopano 29. 1. 2015 na: http://dev.mysql.com/tech-resources/articles/mysql-5.6-rc.html

[27] Works With MariaDB. Dostopano 23. 1. 2015 na: https://mariadb.com/kb/en/mariadb/works-with-mariadb/

[28] The MyISAM Key Cache. Dostopano 29. 1. 2015 na: http://dev.mysql.com/doc/refman/5.6/en/myisam-key-cache.html

[29] Segmented Key Cache. Dostopano 29. 1. 2015 na: https://mariadb.com/kb/en/mariadb/segmented-key-cache/

[30] Aria FAQ. Dostopano 29. 1. 2015 na: https://mariadb.com/kb/en/mariadb/documentation/storage-engines/aria/aria-faq/

[31] Introduction to MySQL Full-Text Search. Dostopano 29. 1. 2015 na: http://www.mysqltutorial.org/introduction-to-mysql-full-text-search.aspx

[32] MyISAM Table Storage Formats. Dostopano 18. 2. 2015 na: http://dev.mysql.com/doc/refman/5.6/en/myisam-table-formats.html

[33] Aria Storage Formats. Dostopano 18. 2. 2015 na: https://mariadb.com/kb/en/mariadb/aria-storage-formats/

[34] Aria Storage Engine. Dostopano 29. 1. 2015 na: https://mariadb.com/kb/en/mariadb/documentation/storage-engines/aria/aria-storage-engine/

[35] (2009) Coronel, C. et al. Database Systems: Design, Implementation and Management. 9th ed. Boston: Cengage Learning, 2009. ISBN 978-0-538-74884-1.

[36] Podatkovne baze. Dostopano 29. 1. 2015 na: http://www.s-sers.mb.edus.si/gradiva/rac/moduli/podatkovne_baze/05_supb/03_datoteka.html

[37] MVCC. Dostopano 3. 2. 2015 na: http://community.actian.com/w/index.php/MVCC

[38] (2012) Schwartz, B. et al. High Performance MySQL: Optimization, Backups, and Replication. 3rd ed. Sebastopol: O'Reilly Media, 2012. ISBN 978-1-449-31428-6.

[39] Percona XtraDB Software. Dostopano 23. 1. 2015 na: http://www.percona.com/software/percona-server/percona-xtradb

[40] List of features available in Percona Server releases. Dostopano 16. 2. 2015 na: http://www.percona.com/doc/percona-server/5.5/ps-versions-comparison.html

[41] TokuDB vs. InnoDB. Dostopano 23. 1. 2015 na: http://www.tokutek.com/tokudb-for-mysql/tokudb-vs-innodb/

[42] Hot Indexing Part I: New Feature. Dostopano 23. 1. 2015 na: http://www.tokutek.com/2011/04/hot-indexing-part-i-new-feature/

[43] Hot Column Addition and Deletion Part I: Performance. Dostopano 23. 1. 2015 na: http://www.tokutek.com/2011/03/hot-column-addition-and-deletion-part-i-performance/

[44] TokuDB: High-Performance Storage Engine for MySQL, MariaDB, and Percona Server. Dostopano 16. 2. 2015 na: http://www.tokutek.com/tokudb-for-mysql/

[45] How to enable TokuDB in MariaDB. Dostopano 23. 1. 2015 na: https://mariadb.com/kb/en/mariadb/documentation/storage-engines/tokudb/how-to-enable-tokudb-in-mariadb/

[46] Virtual Columns. Dostopano 23. 1. 2015 na: https://mariadb.com/kb/en/mariadb/documentation/sql-structure-and-commands/sql-commands/data-definition/create/virtual-columns/

[47] Generated Columns in MySQL 5.7.5. Dostopano 23. 1. 2015 na: http://mysqlserverteam.com/generated-columns-in-mysql-5-7-5/

[48] MySQL virtual columns. Dostopano 23. 1. 2015 na: http://datacharmer.blogspot.com/2008/09/mysql-virtual-columns.html

[49] Dynamic columns. Dostopano 23. 1. 2015 na: https://mariadb.com/kb/en/mariadb/documentation/sql-structure-and-commands/nosql/dynamic-columns/

[50] Index/optimizations in Dynamic Columns. Dostopano 1. 2. 2015 na: https://mariadb.com/kb/en/mariadb/indexoptimizations-in-dynamic-columns/

[51] New Enhancements for InnoDB Memcached. Dostopano 1. 2. 2015 na: https://blogs.oracle.com/mysqlinnodb/entry/new_enhancements_for_innodb_memcached

[52] NoSQL with MySQL. Dostopano 25. 1. 2015 na: http://www.drdobbs.com/database/nosql-with-mysql/240167115

[53] Internals of the InnoDB memcached Plugin. Dostopano 27. 1. 2015 na: http://dev.mysql.com/doc/refman/5.6/en/innodb-memcached-internals.html

[54] Memcached Protocol. Dostopano 1. 2. 2015 na: https://github.com/memcached/memcached/blob/master/doc/protocol.txt

[55] DeNA/HandlerSocket-Plugin-for-MySQL. Dostopano 28. 1. 2015 na: https://github.com/DeNA/HandlerSocket-Plugin-for-MySQL

[56] (2014) Bartholomew, D. MariaDB Cookbook. Birmingham: Packt Publishing, 2014. ISBN 978-1-78328-439-9.

[57] Strežnik po meri. Dostopano 16. 2. 2015 na: http://www.arnes.si/storitve/splet-posta-strezniki/streznik-po-meri.html

[58] Sysbench in Launchpad. Dostopano 23. 1. 2015 na: https://launchpad.net/sysbench

[59] Sakila Sample Database. Dostopano 23. 1. 2015 na: http://dev.mysql.com/doc/sakila/en/

[60] Sakila Sample Database: Usage Examples. Dostopano 23. 1. 2015 na: http://dev.mysql.com/doc/sakila/en/sakila-usage.html

[61] MariaDB-5.3 optimizer benchmark. Dostopano 17. 2. 2015 na: https://mariadb.com/blog/mariadb-53-optimizer-benchmark

[62] Segmented key cache performance. Dostopano 21. 2. 2015 na: https://mariadb.com/kb/en/mariadb/segmented-key-cache-performance/

[63] Optimizer Feature Comparison Matrix. Dostopano 21. 2. 2015 na: https://mariadb.com/kb/en/mariadb/optimizer-feature-comparison-matrix/

Dodatek A

Izvorna koda dodatnih skript

V dodatek je vključena izvorna koda datotek z dodatnimi skriptami in testi za orodje sysbench, s katerimi smo preverjali performančno plat sistemov in funkcionalnosti.

A.1 checksum.lua

```
pathtest = string.match(test, "(.*/)") or ""
dofile(pathtest .. "common.lua")

function thread_init(thread_id)
  set_vars()
end

function event(thread_id)
  local table_name
  table_name = "sbtest".. sb_rand_uniform(1, oltp_tables_count)
  rs = db_query("CHECKSUM TABLE ".. table_name .. ";")
end
```

A.2 charset_convert.lua

```
pathtest = string.match(test, "(.*/)") or ""
dofile(pathtest .. "strings.lua")
```

```
3
4  function thread_init(thread_id)
5    set_vars()
6  end
7
8  function event(thread_id)
9    local table_name
10   table_name = "sbtest".. sb_rand_uniform(1, oltp_tables_count)
11   rs = db_query("ALTER TABLE ".. table_name .. " CONVERT TO CHARACTER SET
         'utf8' COLLATE 'utf8_unicode_ci';")
12 end
```

A.3 innerjoin.lua

```
1  function prepare ()
2    print("All set!")
3    return 0
4  end
5
6  function event(thread_id)
7    rs = db_query("SELECT CONCAT(customer.last_name, ', ',
         customer.first_name) AS customer, address.phone, film.title FROM
         rental INNER JOIN customer ON rental.customer_id =
         customer.customer_id INNER JOIN address ON customer.address_id =
         address.address_id INNER JOIN inventory ON rental.inventory_id =
         inventory.inventory_id INNER JOIN film ON inventory.film_id =
         film.film_id WHERE rental.return_date IS NULL AND rental_date +
         INTERVAL film.rental_duration DAY < CURRENT_DATE()")
8  end
```

A.4 groupby.lua

```
1  function prepare ()
2    print("All set!")
3    return 0
4  end
5
6  function event(thread_id)
```

```
7   rs = db_query("SELECT CONCAT(customer.first_name, ' ',
        customer.last_name) AS name, COUNT(rental.rental_id) AS rentals FROM
         rental, customer WHERE rental.customer_id=customer.customer_id
        GROUP BY customer.customer_id;")
8 end
```

A.5 subquery.lua

```
1 function prepare ()
2   print("All set!")
3   return 0
4 end
5
6 function event(thread_id)
7   rs = db_query("SELECT customer_id, CONCAT(customer.first_name, ' ',
        customer.last_name) AS name FROM customer WHERE customer_id IN (
        SELECT customer_id FROM rental WHERE inventory_id IN (SELECT
        inventory_id FROM inventory WHERE film_id IN (SELECT film_id FROM
        film_category WHERE category_id=14)));")
8 end
```

A.6 fulltext.lua

```
1 function prepare ()
2   print("All set!")
3   return 0
4 end
5
6 function event(thread_id)
7   rs = db_query("SELECT * FROM film_text WHERE MATCH(title, description)
        AGAINST ('dinosaur space')")
8 end
```

A.7 create_predmeti_cela.lua

```
1 local table_name = "predmeti_cela"
2
```

```
3  function create_insert()
4     local i
5     local index_name = "PRIMARY KEY"
6     local query
7
8     print("Creating table '" .. table_name .. "'...")
9     query = [[
10         CREATE TABLE predmeti_cela (
11         id int(11) NOT NULL AUTO_INCREMENT,
12         naziv varchar(30) NOT NULL,
13         leto year(4) DEFAULT NULL,
14         izvor varchar(30) DEFAULT NULL,
15         znamka varchar(30) DEFAULT NULL,
16         barva varchar(40) DEFAULT NULL,
17         material varchar(30) DEFAULT NULL,
18         dimenzije varchar(40) DEFAULT NULL,
19         opis varchar(255) DEFAULT NULL,
20         cena int(4) NOT NULL,
21         PRIMARY KEY (id),
22         FULLTEXT KEY opis (opis)
23         ) ENGINE=Aria DEFAULT CHARSET=utf8
24       ]]
25
26    db_query(query)
27
28    print("Inserting " .. oltp_table_size .. " records into '" ..
          table_name .. "'")
29
30    db_bulk_insert_init("INSERT INTO " .. table_name .. "(naziv, leto,
          izvor, znamka, barva, material, dimenzije, opis, cena) VALUES")
31
32    local naziv
33    local izvor
34    local znamka
35    local barva
36    local material
37    local dimenzije
38    local opis
```

```
39
40    for j = 1,oltp_table_size do
41       naziv = sb_rand_str([[@@@@@@ @@@@@@@@@@ @@@]])
42       izvor = sb_rand_str([[@@@@@@@@@@@@@@@]])
43       znamka = sb_rand_str([[@@@@@@ @@@@@@]])
44       barva = sb_rand_str([[@@@@@@@@@]])
45       material = sb_rand_str([[@@@@@@@@@@@@@]])
46       dimenzije = sb_rand_str([[## @@, # @@]])
47       opis = sb_rand_str([[@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@]])
48
49       db_bulk_insert_next("('" .. naziv .. "', " .. sb_rand(1900, 2005) ..
              ", '".. izvor .."', '" .. znamka .. "', '".. barva .."', '"..
             material .."', '".. dimenzije .."', '".. opis .."', " .. sb_rand
             (15, 450) .. ")")
50    end
51    db_bulk_insert_done()
52
53 end
54
55 function prepare()
56    local query
57    db_connect()
58    create_insert()
59    return 0
60 end
61
62 function cleanup()
63    print("Dropping table '" .. table_name .. "'...")
64    db_query("DROP TABLE ".. table_name )
65 end
```

A.8 create_predmeti_din.lua

```
1  local table_name = "predmeti_cela"
2
3  function create_insert()
4     local i
5     local index_name = "PRIMARY KEY"
```

```
6   local query
7
8   print("Creating table '" .. table_name .. "'...")
9   query = [[
10       CREATE TABLE predmeti_cela (
11       id int(11) NOT NULL AUTO_INCREMENT,
12       naziv varchar(30) NOT NULL,
13       leto year(4) DEFAULT NULL,
14       izvor varchar(30) DEFAULT NULL,
15       znamka varchar(30) DEFAULT NULL,
16       barva varchar(40) DEFAULT NULL,
17       material varchar(30) DEFAULT NULL,
18       dimenzije varchar(40) DEFAULT NULL,
19       opis varchar(255) DEFAULT NULL,
20       cena int(4) NOT NULL,
21       PRIMARY KEY (id),
22       FULLTEXT KEY opis (opis)
23       ) ENGINE=Aria DEFAULT CHARSET=utf8
24     ]]
25
26  db_query(query)
27
28  print("Inserting " .. oltp_table_size .. " records into '" ..
        table_name .. "'")
29
30  db_bulk_insert_init("INSERT INTO " .. table_name .. "(naziv, leto,
        izvor, znamka, barva, material, dimenzije, opis, cena) VALUES")
31
32  local naziv
33  local izvor
34  local znamka
35  local barva
36  local material
37  local dimenzije
38  local opis
39
40  for j = 1,oltp_table_size do
41     naziv = sb_rand_str([[@@@@@@ @@@@@@@@@@ @@@]])
```

```
        izvor = sb_rand_str([[@@@@@@@@@@@@@@]])
        znamka = sb_rand_str([[@@@@@@ @@@@@@]])
        barva = sb_rand_str([[@@@@@@@@]])
        material = sb_rand_str([[@@@@@@@@@@@@]])
        dimenzije = sb_rand_str([[## @@, # @@]])
        opis = sb_rand_str([[@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@]])

        db_bulk_insert_next("('" .. naziv .. "', " .. sb_rand(1900, 2005) ..
             ", '".. izvor .."', '" .. znamka .. "', '".. barva .."', '"..
           material .."', '".. dimenzije .."', '".. opis .."', " .. sb_rand
           (15, 450) .. ")")
    end
    db_bulk_insert_done()

end

function prepare()
    local query
    db_connect()
    create_insert()
    return 0
end

function cleanup()
    print("Dropping table '" .. table_name .. "'...")
    db_query("DROP TABLE ".. table_name )
end
```

A.9 dynamic_cela.lua

```
pathtest = string.match(test, "(.*/)") or ""
dofile(pathtest .. "create_predmeti_cela.lua")

function thread_init(thread_id)
    set_vars()
end

function event(thread_id)
```

```
9     local table_name
10    table_name = "predmeti_cela"
11    rs = db_query("SELECT naziv, leto, barva, opis, cena FROM "..
          table_name .." WHERE id=" .. sb_rand(1, oltp_table_size))
12 end
```

A.10 dynamic_din.lua

```
1  pathtest = string.match(test, "(.*/)") or ""
2  dofile(pathtest .. "create_predmeti_din.lua")
3
4  function thread_init(thread_id)
5     set_vars()
6  end
7
8  function event(thread_id)
9     local table_name
10    table_name = "predmeti_din"
11    rs = db_query("SELECT COLUMN_GET(predmet, 'naziv' as CHAR) as naziv,
          COLUMN_GET(predmet, 'leto' as int) as leto, COLUMN_GET(predmet, '
          barva' as CHAR) as barva, COLUMN_GET(predmet, 'opis' as CHAR) as
          opis, COLUMN_GET(predmet, 'cena' as INT) as cena FROM "..
          table_name .." WHERE id=" .. sb_rand(1, oltp_table_size))
12 end
```

A.11 dynamic_gbp_cela.lua

```
1  pathtest = string.match(test, "(.*/)") or ""
2  dofile(pathtest .. "create_predmeti_cela.lua")
3
4  function thread_init(thread_id)
5     set_vars()
6  end
7
8  function event(thread_id)
9     local table_name
10    table_name = "predmeti_cela"
```

```
11    rs = db_query("SELECT naziv, opis, cena, cena*0.74 as cena_gbp FROM "..
          table_name .." WHERE id=" .. sb_rand(1, oltp_table_size))
12 end
```

A.12 dynamic_gbp_virt.lua

```
1  pathtest = string.match(test, "(.*/)") or ""
2  dofile(pathtest .. "create_predmeti_din.lua")
3
4  function thread_init(thread_id)
5     set_vars()
6  end
7
8  function event(thread_id)
9     local table_name
10    table_name = "predmeti_din"
11    rs = db_query("SELECT naziv, opis, cena, cena_gbp FROM ".. table_name
          .." WHERE id=" .. sb_rand(1, oltp_table_size))
12 end
```